Manual of the SHELL-ATM code

Abstract

This is the user's manual for the SHELL-ATM code.

Contents

1 Introduction to the code **2**
- 1.1 Purpose and basic concepts . . . 2
- 1.2 System requirements . . . 3
- 1.3 Obtaining the code . . . 3
- 1.4 Source tree directory structure . . . 3

2 Using the code **4**
- 2.1 Installing the code for running . . . 4
- 2.2 Tuning the makefiles . . . 4
- 2.3 Choosing the modules . . . 4
- 2.4 Compiling the code . . . 5
- 2.5 Choosing the parameters . . . 5
- 2.6 Choosing the output . . . 7
- 2.7 Launching a (parallel) run . . . 8

3 Analyzing the results **8**
- 3.1 Output files and format . . . 8
- 3.2 Tutorial for analysis with IDL . . . 11
- 3.3 IDL routines . . . 16

4 Programming the code **37**
- 4.1 Types and variables . . . 37
- 4.2 Modules, routines and functions . . . 38
- 4.3 Adding a module . . . 60
- 4.4 Parallelization . . . 61
- 4.5 Bugs . . . 61
- 4.6 Wishlist . . . 61

5 A few tricks: how to... **62**
- 5.1 Continue a run . . . 62
- 5.2 Continue a run with increased number of planes . . . 62

A Authors and contributors **62**

B License agreement **62**

Conventions for this document

- Code and content of files are in `typewriter font`
- Commands to type are `highlighted in light blue`
- File and directory names are `highlighted in light green`

- Variables, types, modules, functions, and routines are `highlighted in light yellow`

1 Introduction to the code

1.1 Purpose and basic concepts

The Shell-Atm code (Buchlin and Velli, 2007; Verdini *et al.*, 2009) is designed to provide fast simulations of MHD turbulence in a volume dominated by a longitudinal magnetic field $\vec{B}_0$. Direct numerical simulations in 3D are indeed much too slow to produce long time series (suitable for a statistical analysis) at sufficient Reynolds number.

Because of the strong uniform $\vec{B}_0$ field, the turbulent, non-linear dynamics occurs essentially in the planes orthogonal to $\vec{B}_0$. The Shell-Atm model consists thus of a set of planes orthogonal to $\vec{B}_0$, in which the non-linear dynamics can be modelled. These planes are piled up along $\vec{B}_0$, and Alfvén waves propagate between them, allowing the transport of energy from the boundaries of the simulation box.

This model includes:

- Fluctuations of magnetic and velocity fields perpendicular to $\vec{B}_0$
- Alfvén wave propagation along $\vec{B}_0$
- Non-linear interactions of counter-propagating Alfvén waves
- Open or reflecting (with forcing by an imposed velocity field) boundary conditions
- Dissipation through diffusivity terms
- Constant (imposed) stratification of density, magnetic field and Alfvén velocity
- Choice of different geometries and over-expansion factor of the flux tube

but it does not include:

- Large-scale velocity and magnetic fields (except the imposed $\vec{B}_0$ field): no wind, no flows
- Non-MHD effects, energetic particles
- Gravity (except for the imposed density stratification)
- Thermodynamics

The code:

- Is fully modularized and allows easy selection of models
- Is parallelized with MPI

The non-linear model for which the code was initially developed is a shell-model: each plane (the basic "brick" of the simulation box) is a shell-model of incompressible MHD turbulence (Gloaguen *et al.*, 1985; Biskamp, 1994; Giuliani and Carbone, 1998; Boffetta *et al.*, 1999; Giuliani *et al.*, 2002). In this kind of models, the Fourier space is divided in concentric shells $S_n = \{\vec{k} \mid \|\vec{k}\| \in [k_n, k_{n+1}]\}$ with $k_n = k_0\lambda^n$, $n = 0, \ldots,$ `nk` $- 1$, and usually $\lambda = 2$. A single complex scalar value u_n represents the velocity increments $|u(x+\ell) - u(x)|$ on scales ℓ for $2\pi/\ell \in S_n$. The same holds for the scalar value b_n representing the magnetic field increments on the same scales ℓ. This one-dimensional model is the magnetohydrodynamic analog of the GOY (Gledzer-Ohkitani-Yamada: Gledzer, 1973; Yamada and Ohkitani, 1987, 1988b,a) shell-model of fluid turbulence, and it can represent the turbulent dynamics in any dimension.

It is governed by the following equation, given in Giuliani and Carbone (1998):

$$\frac{\mathrm{d}Z_n^s}{\mathrm{d}t} = -k_n^2(\nu^+ Z_n^s + \nu^- Z_n^{-s}) + ik_n T_n^{s*} + f_n^s \tag{1}$$

where $Z_n^s = v_n + sb_n$ are the Elsässer variables, $s = \pm$, $\nu^s = (\nu + s\eta)/2$ are combinations of kinematic viscosity and resistivity, f_n^s are external driving forces, and T_n^s is the non-linear term.

By imposing that this non-linear term conserves the total pseudo-energies $E^s = \sum_n |Z_n^s|^2$ (and thus the energy $E = E^+ + E^-$ and the cross-helicity $h_C = E^+ - E^-$) and a third invariant $H_K^\alpha = \sum_n \mathrm{sign}(\delta - 1)^n k_n^\alpha |v_n|^2$ which depends on the dimensionality of the MHD system to model, and by assuming that the important non-linear interactions are between triads of neighboring modes, we get the following form for T_n^s given in Giuliani and Carbone (1998):

$$\begin{aligned} T_n^s = & \frac{\delta + \delta_m}{2} Z_{n+1}^s Z_{n+2}^{-s} + \frac{2 - \delta - \delta_2}{2} Z_{n+1}^{-s} Z_{n+2}^s + \\ & \frac{\delta_m - \delta}{2\lambda} Z_{n+1}^s Z_{n-1}^{-s} - \frac{\delta + \delta_m}{2\lambda} Z_{n+1}^{-s} Z_{n-1}^s - \\ & \frac{\delta_m - \delta}{2\lambda^2} Z_{n-1}^s Z_{n-2}^{-s} - \frac{2 - \delta - \delta_m}{2\lambda^2} Z_{n-1}^{-s} Z_{n-2}^s \end{aligned} \tag{2}$$

1.2 System requirements

The following is needed to run the Shell-Atm code:

- Fortran 90 compatible compiler
- C compiler
- Unix system or subsystem (tested on: GNU/Linux)
- MPI parallel library (*e.g.* MPICH, OpenMPI)

The following is useful:

- Perl for generating automatically parameters sets and documentation
- IDL for visualizing the results with the provided routines

1.3 Obtaining the code

The code can currently be obtained only by writing to the authors. Once you have the tarball, put it in a convenient directory and unpack it using the following command:

```
unix>  tar zxvf shell-atm.tgz
```

The code is also under the SVN (SubVersioN) revision control system. Please contact the authors for SVN access.

1.4 Source tree directory structure

The source tree directory is organized as follows:

`config/` configuration files specific to some platforms or models

`doc/` documentation, HTML files for source code documentation, and LaTeX source of the manual

`idl/` IDL analysis routines

`runs/` directory where the code will be installed for the runs

`src/` source code

`sys/` scripts for handling the code

2 Using the code

2.1 Installing the code for running

To install[1] the code use the following command:
`make install DIR=myrundir MODEL=mymodel PLATFORM=myplatform/mycomputer`
where:

- `myrundir/` is the subdirectory of the directory `runs` where the code will be installed
- `mymodel/` is the subdirectory of the directory `config/model` where default model-specific `Makefile` (`Makefile.model`, containing the names of the modules used for the model) and parameters (`params.txt`, `param_o.txt`) can be found. If this is not provided, `default/` is used. If `mymodel/` does not contain parameters files, they are copied from `src/` instead.
- `myplatform/mycomputer` is the subdirectory of the directory `config/platform` where the default platform-specific `Makefile` can be found. `mycomputer` corresponds to the host name of computers on which the code has been run. `myplatform` corresponds to the system (Linux, Darwin...). If `myplatform/mycomputer` is not provided, it is determined from the result of the Unix commands `uname` and `hostname`.

New directories of the type `config/mymodel/` and `config/myplatform/mycomputer/` should be created if needed.

2.2 Tuning the makefiles

Go to the directory where the code was installed (`runs/myrundir` if `DIR` was set to `myrundir` during the installation) and edit the `Makefile.platform` to adapt the variables to your system (this shouldn't be necessary if the right platform has been selected when installing the code):

`MPIFC` Fortran 90 or 95 compiler for MPI parallel programs (usually this is a script provided by the MPI library)

`MPIFOPTS` Options for the Fortran compiler

`CC` C compiler

2.3 Choosing the modules

In the file `Makefile.modules` of the installation directory, variables allow to conveniently select the modules among the following alternatives (this shouldn't be necessary if the right model has been chosen when installing the code):

`ATMOSPHERE` Type of atmosphere geometry:

- `cart` Cartesian atmosphere, with stratification
- `dmit` atmosphere according to Dmitruk 04 [***ref***]
- `spher` spherical atmosphere, with stratification
- `stratin` stratified atmosphere, with Alfvén speed profile. The `fexpand` module is then not used. The Alfvén speed is read in `in_va` and the mass density profile is read in file `in_rho`
- `unif` uniform atmosphere

`FEXPAND` Type of expansion factor of the flux tube (in addition to the expansion possibly resulting from the geometry):

- `none` no over-expansion

[1] Warning: As module alternatives need to be chosen in the `Makefile` (Sec. 2.2), this only installs source files, not the compiled code.

`MJ` over-expansion according to Munro-Jackson (1981) [***ref***]

`IODATA` Formatted or unformatted output:

`formatted` formatted output

`unformatted` unformatted output

This affects also the input in `read_simstate`

`NONLIN` Model used for the non-linear interactions in planes

`goy` 2D GOY shell-models from Giuliani and Carbone (1998)

`goyb` 2D GOY shell-models from Giuliani and Carbone (1998) with Alfvén effect term from Biskamp (1994)

`nlsabra` 2D version of non-local Sabra shell-model (Plunian and Stepanov, 2007)

`none` No non-linear interactions

`UNIT` Units of variables of the model (length, time, mass). These units will set the adimensionalization for all variables in the code, which includes output and parameters input. In this documentation they will be hereafter referred to as ([L], [T], [M]).

`open` Variables adapted for open-field solar corona simulations (solar radius, second, gramme)

`CGS` Variables in CGS units (centimeter, second, gramme)

The modules and module alternatives will be detailed in Sec. 4.2.

2.4 Compiling the code

Use the command `make` to produce the executable `shell-atm`.

2.5 Choosing the parameters

Parameters files are text files with odd lines containing comments and even lines containing the value for the variable describe in the comment. When parameters have a dimension, this dimension is assumed to be consistent with the units given in the alternative `UNIT` of module `unit`.

2.5.1 In `src/params.txt`

The content of this file sets the value of these parameters of the simulation (the names correspond to the names of variables in the code):

`p%nmin` Index $n_\perp$ of first shell

`p%nmax` Index $n_\perp$ of last shell

`p%nz` Number n_z of planes[2] (piled up in the direction of $\vec{B}_0$).

`p%k0` Perpendicular wavenumber, corresponding to $n_\perp = 0$ (unit: $[\mathrm{L}]^{-1}$)

`p%lambda` Ratio between wavenumber of successive perpendicular modes (should be 2)

`p%nu` Viscosity (unit: $[\mathrm{L}]^2 \cdot [\mathrm{T}]^{-1}$)

`p%eta` Magnetic diffusivity (unit: $[\mathrm{L}]^2 \cdot [\mathrm{T}]^{-1}$)

`p%eps` δ parameter of the shell-model (Giuliani and Carbone, 1998)

[2] Note that in the code, `p%nz` is the number of planes *for the current processor*, and that the total number of planes is called `p%tnz`.

`p%epsm` δ_m parameter of the shell-model (Giuliani and Carbone, 1998)

`p%b0` Alfvén velocity parameter (exact meaning depends on module `atmosphere`) (unit: $[\mathrm{L}]\cdot[\mathrm{T}]^{-1}$)

`p%dz` Distance between planes (unit: [L])

`p%nt`
- if positive, number of time steps to run
- if negative, opposite of time when the simulation should stop (unit: [T])

`p%delta_t` Initial time step. May be deprecated soon, as `check_time_scales` is called before the first cycle of the main loop (unit: [T])

`p%continue` if 1, continue from the simulation state saved in `simstate.dat` by a previous run. If 0, start a new run with initial conditions from the routine `init_shells` of the module `initialize` (this feature may be deprecated soon as it is easy to build an initial simulation state by external programs, which would be more flexible)

`p%initamp` Amplitude of initial field, as interpreted by `init_shells` (unit: $[\mathrm{L}]\cdot[\mathrm{T}]^{-1}$)

`p%initslope` Slope of initial fields, as interpreted by `init_shells`

`p%forcamp` Amplitude of forcing, used in `force` (unit: $[\mathrm{L}]\cdot[\mathrm{T}]^{-1}$)

`p%tstar` Correlation time of forcing, used in `force` (unit: [T])

`p%rho0` Mass density, as interpreted by the function `rho_det` of the module `atmosphere` (unit: $[\mathrm{M}]\cdot[\mathrm{L}]^{-3}$)

`p%h` Scale height, as interpreted by the functions `rho_det`, `rhodl_det` and `cs_det` of the module `atmosphere` (unit: [L])

Generating parameters from meta-parameters (optional) It is possible to use the Perl script `sys/metaparams.pl` (which can be called by `make params.txt` in `src/`) to generate automatically `params.txt` from a file `mparams.txt` containing "meta-parameters[3]", or parameters which would have a more direct physical meaning than the parameters contained in `params.txt`:

`continue` Same as `continue` from `params.txt`

`a` Simulation box aspect ratio (length divided by width)

`nperp` Number `nk` of shells

`nz` Number n_z of planes (piled up in the direction of $\vec{B}_0$)

`nt`
- if positive, number of time steps to run
- if negative, opposite of time when the simulation should stop (unit: [T])

`modeamp` Target amplitude of the modes at the largest scales, assuming a Kolmogorov spectrum down to the dissipation scales

`forcamp` Amplitude of forcing, used in `force` (unit: $[\mathrm{L}]\cdot[\mathrm{T}]^{-1}$)

However, this script is currently not maintained and may give files which are incompatible with the current version of the code. New meta-parameters should be added, corresponding to new parameters of the simulation.

[3] The names of the meta-parameters given here do not correspond to variable names in the code.

2.5.2 In file `src/paramexp.txt`

The content of this file sets the value of these parameters:

`p%pexp%x0` Position of the first plane of the simulation box (unit: [L]). This allows for example to have distances from the Sun center instead of altitudes

`p%pexp%sigma` Width of expansion (unit: [L])

`p%pexp%fmax` Maximum expansion (unit: [L])

`p%pexp%x1n` Position of the expansion (unit: [L])

[***These are read but don't seem to be used yet***]

2.5.3 Files `in_va` and `in_rho`

These files contain the profiles of Alfvén speed and mass density read when module `atmosphere-stratin` is used.

2.6 Choosing the output

The content of the file `src/param_o.txt` sets the following parameters:

`outflags%period` Period for main output (output of time series of integrated quantities, at times denoted "A" in Sec. 3.1):

- if positive: number (integer) of time steps between each output
- if negative: opposite of time (floating point) between each output, which is then called `outflags%periodt`

`outflags%specperiod` Period for the output of non-integrated quantities (at times denoted "B" in Sec. 3.1), such as spectra, raw fields, simulation state (including raw fields):

- if positive: number (integer) of time steps between each output
- if negative: opposite of time (floating point) between each output, which is then called `outflags%specperiodt`

Spectra are averaged on some time steps between the current output and the last output.

`outflags%sfunskip` Number of time steps to skip before beginning to evaluate the structure functions (deprecated)

`outflags%sfunnb` Number of structure functions to compute

The next even lines contain 1 or 0, telling whether the following Boolean variables should true or false, and then whether some output files should be written or not[4]:

- `outflags%time`
- `outflags%taunl`
- `outflags%enmode02`
- `outflags%enmode08`
- `outflags%enmode14`
- `outflags%enmode20`
- `outflags%entotu`
- `outflags%entotb`
- `outflags%dissu`
- `outflags%dissb`
- `outflags%forcp`
- `outflags%crosshel`
- `outflags%maghel`
- `outflags%specu`
- `outflags%specb`
- `outflags%sfunu`
- `outflags%sfunb`
- `outflags%rawzp`
- `outflags%rawzm` (deprecated)
- `outflags%profva`

[4] See Sec. 3.1 for the details of output files and the association between these Boolean variables and the files that are output.

- `outflags%profvad`
- `outflags%rad`
- `outflags%profrho`
- `outflags%profaexp`

Others are not used yet and the meaning indicated in the comments of `src/param_o.txt` may change.

2.7 Launching a (parallel) run

2.7.1 For an interactive run

Use `make NP=nb_processors mpirun`.

2.7.2 For a run with the SLURM (Simple Linux Utility for Resource Management) queueing system

You can directly use `srun -c nb_processors make NP=nb_processors mpirun`, or `make NP=nb_processors mpirun` inside a shell opened by `salloc`.

2.7.3 For a run with the LSF (Load Sharing Facility) queueing system

Edit the file `mpi_lsf.txt` and specify the number of processors, the maximum wall-clock time, the files where standard output and errors should be logged, and the commands to run.

`mpi_lsf.txt` could for example look like this:

```
#!/bin/tcsh
# Number of processors and wall-clock time
#BSUB -n 20 -W 10
# MPI (mandatory)
#BSUB -a mpich_gm
# output and error logs
#BSUB -o output.log -e error.log

date
mpirun.lsf ./shell-atm
```

Then run `make mpirunlsf`.

2.7.4 For a run with the PBS queueing system

Edit the file `mpi_pbs.txt` and specify the number of processors, the maximum wall-clock time, the files where standard output and errors should be logged, and the commands to run.

Then run `make mpirunpbs`.

3 Analyzing the results

3.1 Output files and format

Data are output as formatted numbers in files `out_*`. Flags of the `outflags` structure (of type `saveflags`) determine whether a file is output or not.

As files for output of time series of integrated quantities remain open during the whole simulation, one should choose carefully the associated file-handles, which are also listed below.

Table 1: Output and input files and associated parameters.

File	FH	Flag	W	F	Description
		Output of times relevant for interpreting other output files			
out_time	10	time	A	A	Time of output of integrated quantities
out_taunl	101	taunl	A	A	Non-linear time scale
out_time2i	102	rawzp	D′	A	Indices of times when out_simstate_* are output
out_time2	103	spec[ub], rawzp	B	A	Time of output of non-integrated quantities
		Output of integrated quantities			
out_ent02	11	eninmode02	A	A	Total energy in mode $n_\perp = 2$
out_ent08	12	eninmode08	A	A	Total energy in mode $n_\perp = 8$
out_ent14	13	eninmode14	A	A	Total energy in mode $n_\perp = 14$
out_ent20	14	eninmode20	A	A	Total energy in mode $n_\perp = 20$
out_enu	20	entotu	A	A	Total kinetic energy
out_enb	21	entotb	A	A	Total magnetic energy
out_dissu	22	dissu	A	A	Kinetic energy dissipation power
out_dissb	23	dissu	A	A	Magnetic energy dissipation power
out_forcp	24	forcp	A	A	Power of the forcing at the boundaries
out_crosshel	26	crosshel	A	A	Total cross helicity
out_maghel	27	maghel	A	A	Total "magnetic helicity" (or third conserved quantity of shell-models)
		Output of spectra and structure functions			
out_specu	30	specu	B	B	Kinetic energy perpendicular spectra, summed on all planes, averaged in time since last output
out_specb	31	specb	B	B	Magnetic energy perpendicular spectra, summed on all planes, averaged in time since last output
out_sfunu	35	sfunu	C	C	Velocity field structure functions
out_sfunb	36	sfunb	C	C	Magnetic field structure functions
		Output of simulation state			
simstate.dat	91	N/A	D	D	Simulation state, for restarting simulation
out_simstate_nnn	91	rawzp	D′	D	Simulation state, for computing quantities and statistics needing the whole raw fields.
		Output of atmospheric profiles as a function of altitude			
out_va	50	profva	E	E	Alfvén velocity
out_vad	51	profva	E	E	Alfvén velocity gradient
out_rho	52	profrho	E	E	Density
out_rad	53	profrad	E	E	Distance from the base of the simulation box
out_aexp	54	profaexp	E	E	Expansion factor
out_profheat	55	profheat	C	E	Profile of the dissipation power per unit length, averaged over the whole simulation
		Input of parameters			
params.txt	90	N/A	F	F	Parameters of the simulation
paramexp.txt	90	N/A	F	F′	Parameters of flux tube expansion
param_o.txt	92	N/A	F	F″	Parameters of output
in_va	99	N/A	F	E	Alfvén speed profile
in_rho	99	N/A	F	E	Mass density profile

Meaning of codes in the column "W" (when data is output):

- A: Depending on the sign of the value provided for `outflags%period` in `param_o.txt`:
  - if positive (and integer): every `outflags%period` time steps
  - if negative (and floating point): every `outflags%periodt` = −`outflags%period` units of time [T]
- B: Depending on the sign of the value provided for `outflags%specperiod` in `param_o.txt`:
  - if positive (and integer): every `outflags%specperiod` time steps
  - if negative (and floating point): every `outflags%specperiodt` = −`outflags%specperiod` units of time [T]
- C: At the end of the simulation (but is an average evaluated over the whole simulation, excluding the `sfunskip` first time steps for the structure functions)
- D: Every time `write_simstate` is called with `simstatefile=.TRUE.` option, *i.e.* every $10^7/n_z$ time steps, and at the end of the simulation. This overwrites the previous version of the file.
- D': Every time `write_simstate` is called without `simstatefile=.TRUE.` option, which should correspond to code "B" above. `nnn` is a 0-padded 10-digit number indicating the timestep of output (these are also listed in the `out_time2i` file)
- E: Every time `grid_assign` is called, *i.e.* once at the beginning of the simulation
- F: Every time `read_parameters` is called, *i.e.* once at the beginning of the simulation
- G: Every time `read_output_options` is called, *i.e.* once at the beginning of the simulation

Meaning of codes in the column "F" (format of the file):

- A (typically: time series of integrated quantities): list of values of the time series. The number of output values is stored in `nrecen` in `simstate.dat`
- B (spectra): serialized 3D array of real values (squared amplitudes of modes), of dimension (`nk`, `tnz`, `nrecsp`). `nrecsp` is read in `simstate.dat`.
- C (structure functions): serialized 3D array of real values, of dimension (`nk`, `tnz`, `sfunnb`).
- D (simulation state):
  - Parameters of the current run: `nmin`, `nmax`, `tnz` (number of planes, summed over all processors), `k0`, `lambda`, `nu`, `eta`, `eps`, `epsm`, `b0`, `ti` (number of time steps performed), `delta_t`
  - Number of output of each type: `nrecen`, `sfunnb`, `nrecsp`
  - Serialized 3D array of dimension (2, `nk`, `tnz`) of values (real and imaginary, for each shell, for each plane) of fields for `zp`
  - Same for `zm`
  - Current time
  - Other parameters: `tstar` (correlation time of forcing), `nkforce` (number of modes on which the forcing is made)
  - Current random coefficients of forcing (to be able to continue a simulation with no discontinuity in the forcing): this is a serialized 4D array of dimension (2, 2, `nkforce`, 2) (real and imaginary, for both components, for all forced modes, for both boundaries)
  - Parameters of atmosphere: `h`, `rho0`
- E (profiles of atmosphere): 1D array of real values of field as a function of grid point in the parallel direction
- F, F', F'': see section 2.5

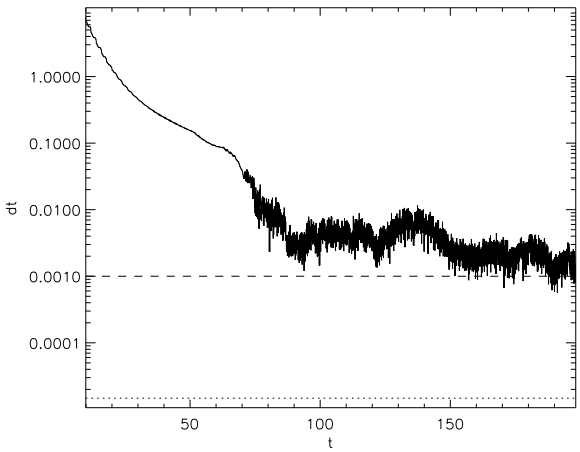
3.2 Tutorial for analysis with IDL

Here is an example of an IDL session where data output by the model is analyzed. First, we can get the parameters of the run with `sa_params_get`. By default, the file `simstate.dat` of the current directory is used, but other simulation state outputs (`out_simstate_nnn`) can be used if the name of the file is passed as an argument of `sa_params_get`.

```
Idl> p = sa_params_get ()
% Compiled module: SA_PARAMS_GET.
% Compiled module: SA_PARAMS__DEFINE.
Idl> help, p, /struct
** Structure SA_PARAMS, 19 tags, length=120, data length=120:
   NMIN            INT               0
   NMAX            INT              17
   NZ              LONG           1000
   K0              DOUBLE        62.831853
   LAMBDA          DOUBLE         2.0000000
   NU              DOUBLE    1.0000000e-09
   ETA             DOUBLE    1.0000000e-09
   EPS             DOUBLE         1.2500000
   EPSM            DOUBLE        -0.33333333
   B0              DOUBLE         1.0000000
   DZ              DOUBLE      0.0010000000
   NT              LONG         6400000
   NRECEN          LONG           18824
   NRECSP          LONG             189
   NRECSF          LONG               0
   TIME            DOUBLE       198.18528
   TSTAR           DOUBLE       200.31240
   RHO0            DOUBLE         1.0000000
   H0              DOUBLE         1.0000000
```

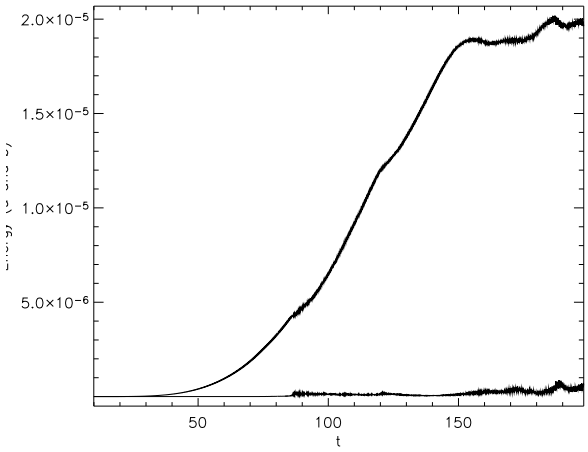
Then with `sa_plot_dt` we can plot the evolution of the smallest time scales of the model ($\tau_{A,\min}$ and $\tau_{\nu,\min}$ which are constant, and $\tau_{\mathrm{NL},\min}$). `minmax` allows to plot only the given number of useful points on the graph, to get light graphic files even with long time series.

```
Idl> sa_plot_dt, /tau, minmax=1000
% Compiled module: SA_PLOT_DT.
% Compiled module: READARRAY.
% Compiled module: CONGRID.
```



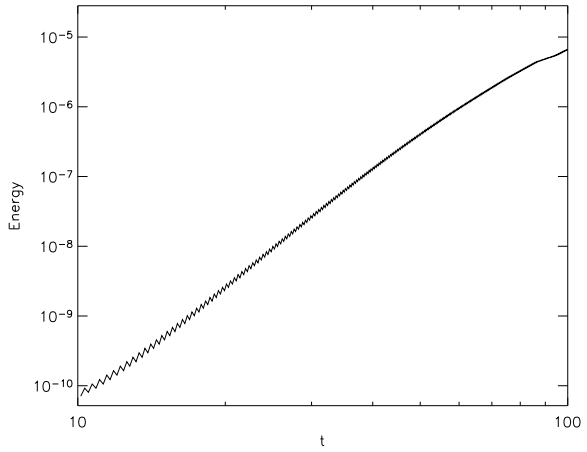
We can also plot with `sa_plot_energy` the evolution of the energies (kinetic and magnetic) in the box, in units consistent with the ones chosen in the module `unit`:

```
Idl> sa_plot_energy, minmax=1000
% Compiled module: SA_PLOT_ENERGY.
```



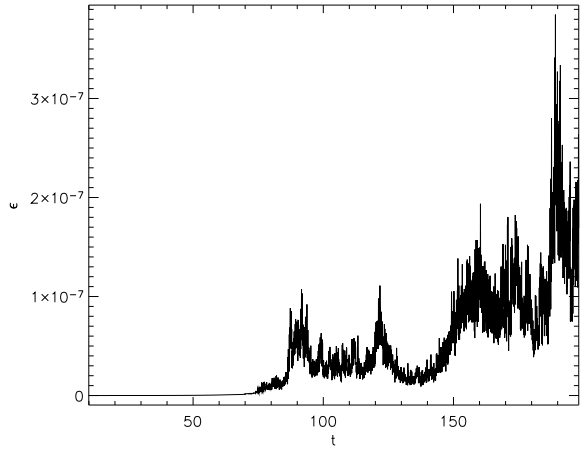
We can choose to plot the total energy in logarithmic scales and on a restricted time interval:

```
Idl> sa_plot_energy, /tot, /xlog, /ylog, xrange=[10, 100], minmax=500
```

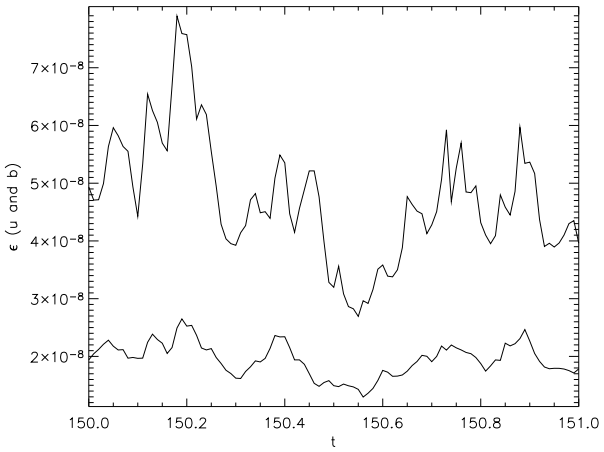


`sa_plot_de` plots the energy dissipations in the box, with a syntax similar to the one of `sa_plot_energy`:

```
Idl> sa_plot_de, /tot, minmax=1000
% Compiled module: SA_PLOT_DE.
```

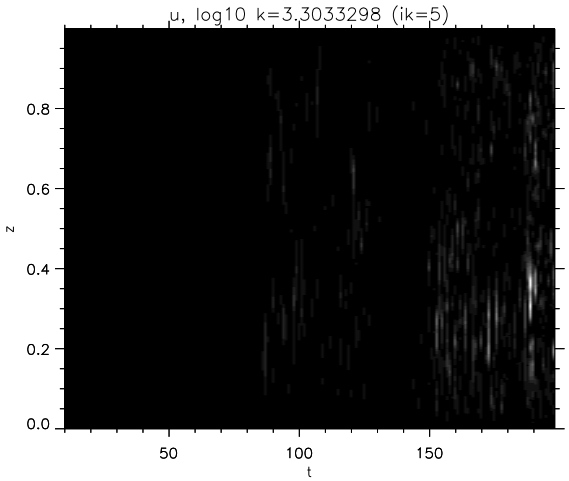


```
Idl> sa_plot_de, xrange=[150,151]
```



Then we can plot 2D cuts of the data (which depend on `k`, `z` and `t`). First, with `sa_plot_cutk` a cut of the kinetic energy (`/uu`)as a function of `z` and `t` (for a given `k`):

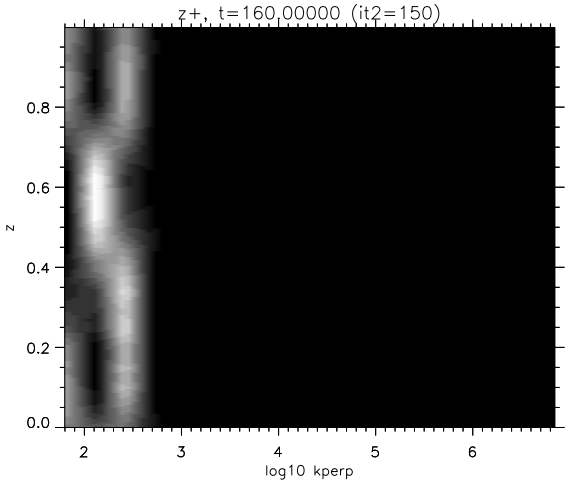
```
Idl> sa_plot_cutk, 5, /uu, /noi
% Compiled module: SA_PLOT_CUTK.
% Compiled module: SA_PLOT_CUTS_TIDYOPTIONS.
% Compiled module: UNIQ.
% Compiled module: READARRAY_S.
% Compiled module: SA_SHELLS_TO_FIELDS.
% Compiled module: SA_PLOT_CUTS_NORMALIZE.
% Compiled module: STRING_NS.
% Compiled module: PLOT_IMAGE_EB.
Range:           0.0000000      1.0382235e-06
```



`/noi` tells that the image is plot with no interpolation (this would lead to a distortion of the image if an axis would have points which are not equally spaced — it can be the case of `t`).

With `sa_plot_cutt` a cut of the energy of z^+ (`/zp`) as a function of `k` and `z` (for a given `t`):

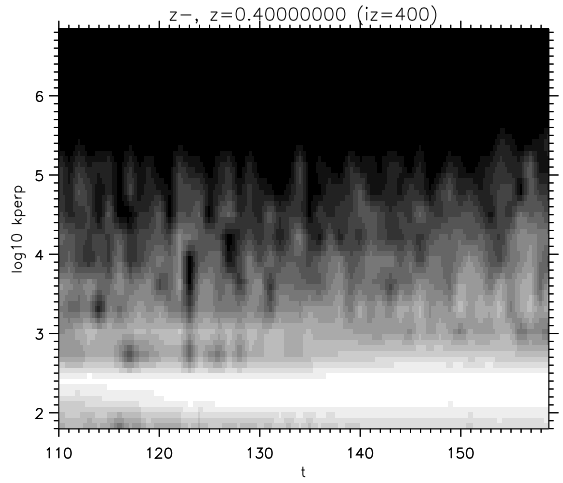
```
Idl> sa_plot_cutt, 150, /zp
% Compiled module: SA_PLOT_CUTT.
% Compiled module: INTERPOL2D.
% Compiled module: INTERPOL.
Range:       5.9676350e-59      1.4663594e-05
```



The range of the computed color bar is given in output.

With `sa_plot_cutz` a cut of the energy of z^- (`/zm`) as a function of `k` and `t` (for a given `z`):

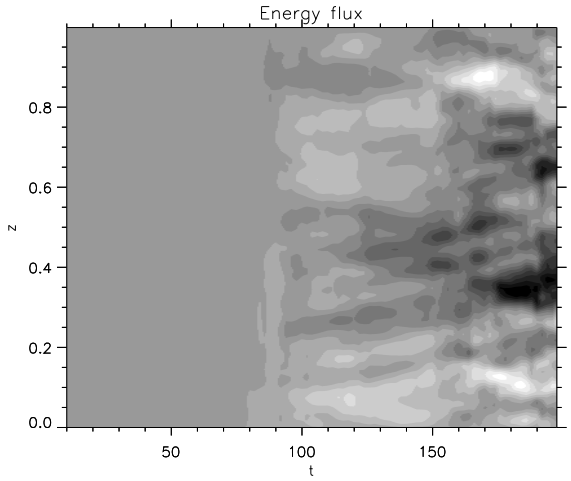
```
Idl> sa_plot_cutz, 400, /zm, time=100+indgen(50), /log, range=[-10,-5]
% Compiled module: SA_PLOT_CUTZ.
Range:       0  255
```



Here only some times (given in number of outputs of `out_simstate_nnn`) are selected, the color range is logarithmic and limited to the given boundaries.

A similar syntax exists for `sa_plot_enflux`, which plots the energy flux as a function of time `t` and position `z`:

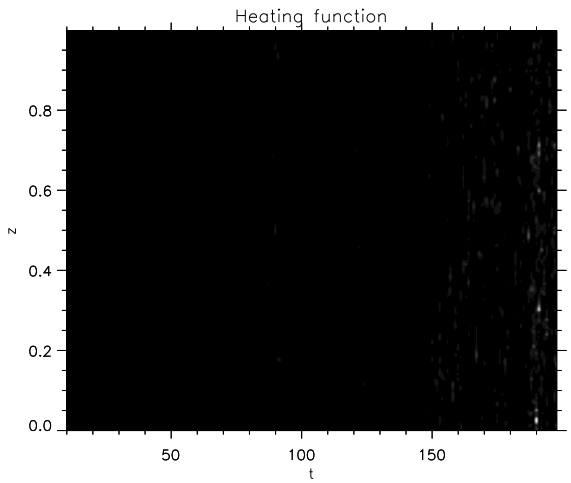
```
Idl> sa_plot_enflux, time=2*indgen(100)
% Compiled module: SA_PLOT_ENFLUX.
Range:      -3.4820471e-06     2.6484825e-06
```



Here we have selected only the even output times so as to remove the periodicity due to the wavepackets travel time along the simulation box.

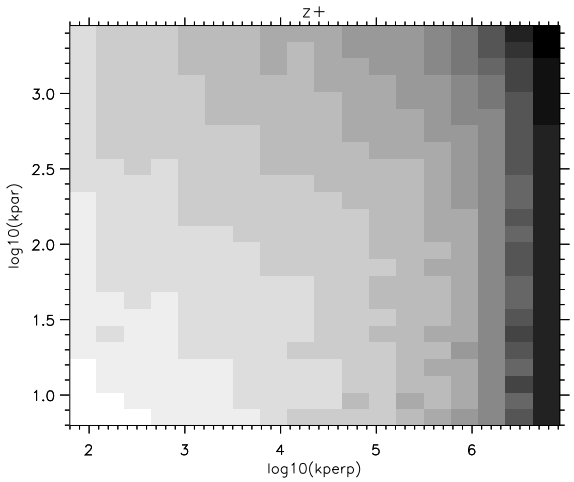
`sa_plot_heatfunc` plots the heating function (lineic power of energy dissipation as a function of time and position):

```
Idl> sa_plot_heatfunc
% Compiled module: SA_PLOT_HEATFUNC.
Range:       5.8631139e-14     6.7019078e-06
```



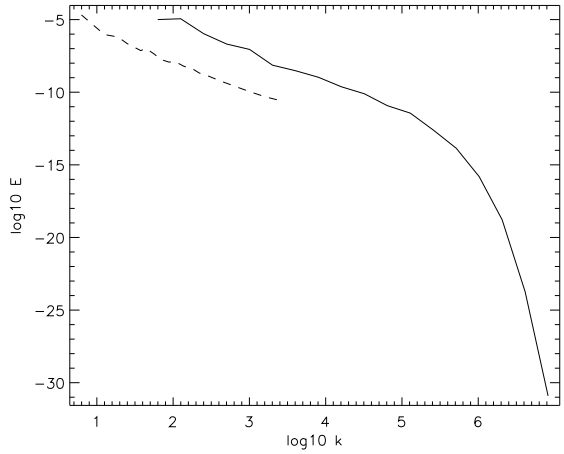
From `simstate.dat` (or another simulation state file given as argument) `sa_plot_spectrum2d` plots the 2D spectrum of a field:

```
Idl> sa_plot_spectrum2d, /zp
% Compiled module: SA_PLOT_SPECTRUM2D.
% Compiled module: LOGSMOOTH.
```



And `sa_plot_spectrumpp` plots the parallel and perpendicular 1D spectra on the same plot:

```
Idl> sa_plot_spectrumpp, /zm
% Compiled module: SA_PLOT_SPECTRUMPP.
```



3.3 IDL routines

The following routines are provided in the `idl/` directory to help analyzing the results of the runs, along with useful ancillary routines. The analysis routines for the SHELL-ATM model have names beginning with `sa_`. They should all be compatible with the data coming from the older shell-loop model, if used with the keyword parameter `atm=0`. Routines with names beginning with `sl_` are wrappers around the corresponding `sa_` routines with `atm=0`.

3.3.1 sa_data_merge

```
NAME:
sa_data_merge

PURPOSE:
Merge data output by successive simulations of the shell-atm model

CALLING SEQUENCE:
sa_data_merge, dirs_in, dir_out, atm=atm

INPUTS:
```

```
dirs_in: list of directories to merge
dir_out: directory where to put merged data

OPTIONAL INPUTS:
atm: if 1 (default), is for shell-atm, if 0, is for shell-loop

SIDE EFFECTS:
Creates and fills dir_out

RESTRICTIONS:
. dirs_in should contain successive parts of the same simulation.
. Unix-only (see spawn)

LANGUAGE:
IDL

PROJECT:
shell IAS

TODO:
Merge also out_simstate_nnn and time2i (need to remap time2i and nnn?)

AUTHORS:
Eric Buchlin, http://eric.buchlin.org/

MODIFICATION HISTORY:
21 Apr 2004 EB Creation
07 Nov 2005 EB Adapted to shell-atm
```

3.3.2 sa_movie

```
NAME:
sa_movie

PURPOSE:
Show a movie or save an AVI file of data output by shell-atm

CALLING SEQUENCE:
sa_movie, sf, file="file.avi", time=time, atm=atm

OPTIONAL INPUT:
sf: file simstate.dat (default)
atm: if 1 (default), is for shell-atm, if 0, is for shell-loop
/uu, /bb, /zp, /zm: plot u, b, zp or zm
range: value range for color scale

KEYWORD PARAMETERS:
file: AVI file to write
time: time indices to use

OUTPUT:


KEYWORD OUTPUT:
```

```
SIDE EFFECTS:
Show a movie or save a AVI file

LANGUAGE:
IDL

PROJECT:
shell IAS

AUTHORS:
Eric Buchlin, eric@arcetri.astro.it

MODIFICATION HISTORY:
13 Dec 2002 EB Creation
07 Nov 2005 EB Adapted to shell-atm
```

3.3.3 sa_params_get

```
NAME:
sa_params_get

PURPOSE:
Get parameters used for a shell_loop simulation

CALLING SEQUENCE:
p = sa_params_get ("simstate.dat", shells=shells, time=time, forc=forc,
atm=atm, /unformatted)

INPUTS:
"simstate.dat" : simulation state file, from which parameters will be
extracted

KEYWORD PARAMETERS:
/unformatted: file is "F77" unformatted (binary)
atm: if 1 (default), is for shell-atm, if 0, is for shell-loop

OUTPUT:
p : sa_params structure containing the parameters

OPTIONAL OUTPUTS:
shells: last shell values (in all planes). Dimensions: (kperp, z,
z+/zm)
forc: forcing coefficients
time: the time of the simulation. Deprecated; use p.time instead

LANGUAGE:
IDL

PROJECT:
shell-loop, shell-atm

AUTHORS:
Eric Buchlin, eric@arcetri.astro.it
Andrea Verdini, verdini@arcetri.astro.it

MODIFICATION HISTORY:
```

```
13 Dec 2002 EB Creation
08 Nov 2003 EB Added shells keyword
17 Jun 2005 EB Added support for p.time
10 Oct 2005 AV/EB Added support for shell_atm
18 Nov 2005 EB/AV Added /unformatted keyword
17 Feb 2006 EB Added output of forcing coefficients
```

3.3.4 sa_params_print

```
NAME:
sa_params_print

PURPOSE:
Print parameters used for a shell_loop simulation

CALLING SEQUENCE:
sa_params_print, "simstate.dat", p=p, atm=atm

OPTIONAL INPUT:
"simstate.dat" : simulation state file, from which parameters will be
extracted. If not present, data in keyword parameter p is used. If not
present, the file "simstate.dat" is used.

KEYWORD PARAMETERS:
p: parameters data
atm: if 1 (default), is for shell-atm, if 0, is for shell-loop

LANGUAGE:
IDL

PROJECT:
shell-loop, shell-atm

AUTHORS:
Eric Buchlin, eric@arcetri.astro.it
Andrea Verdini, verdini@arcetri.astro.it

MODIFICATION HISTORY:
05 Feb 2003 EB Creation
10 Oct 2005 EB/AV Added support of shell-atm
```

3.3.5 sa_params_write

```
NAME:
sa_params_write

PURPOSE:
Write parameters for a shell_loop simulation in a fake simstate.dat
file that can be used for data analysis or to start a simulation

CALLING SEQUENCE:
sa_params_write, p, "simstate.dat", shells=shells, time=time,
forc=forc, atm=atm

INPUTS:
p (of type sa_params): parameters
```

```
"simstate.dat" : simulation state file

OPTIONAL INPUT:
shells: values of shells (real array, with real and imaginary parts
of the complex values of the shells). Default is an array of zeroes.
time: the time of the simulation. Deprecated; use p.time instead
(but time overrides p.time)
forc: coefficients of forcing
(complex, dimensions (2[coefficients], nkforce, 2[boundaries]))
atm: if 1 (default), is for shell-atm, if 0, is for shell-loop

OUTPUT:
p : sa_params structure containing the parameters

OPTIONAL OUTPUTS:
shells: last shell values (in all planes)

RESTRICTIONS:
Assumes that nkforce=3 if forc not provided

LANGUAGE:
IDL

PROJECT:
shell-loop, shell-atm

AUTHORS:
Eric Buchlin, eric@arcetri.astro.it

MODIFICATION HISTORY:
22 Apr 2004 EB Creation
07 Jun 2005 EB Support for p.time. Format for double precision output
07 Nov 2005 EB Adapted for shell-atm
17 Feb 2006 EB Added input of forcing coefficients
```

3.3.6 sa_shells_to_fields

```
NAME:
sa_shells_to_fields

PURPOSE:
Transforms shell values array into fields and energies. Default is
to get energies per unit length, but energies per unit mass or per
unit volume can also be obtained.

CALLING SEQUENCE:
s = sa_shells_to_fields (shells, /uu, /bb, /zp, /zm, /energy,
rho=rho, aexp=aexp, k0=k0, /massic,
/volumic, atm = atm)

INPUTS:
shells: array of complex shell values,
3 dimensions:
- first dimensions of the array can be anything
- second dimension: must correspond to the position on
the loop if computing the energies
```

```
- last dimension: 0 is zp, 1 is zm

KEYWORD PARAMETERS:
/uu, /bb, /zp, /zm (one of these is needed): field to compute
/energy: compute lineic energy in field instead of returning the
amplitudes of the field
rho: mass density of plasma
aexp: over-expansion factor (in area): the width of the loop at a
given position is the width deduced from k0, multiplied by square
root of aexp at this position
k0: k0 wavenumber of simulation
/massic:  get energy per unit mass instead of energy per unit length
/volumic: get energy per unit volume instead of energy per unit length
atm: if 1 (default), is for shell-atm, if 0, is for shell-loop

OUTPUTS:
Field or energy of field

SIDE EFFECTS:
None

PROJECT:
shell-loop, shell-atm

AUTHORS:
Eric Buchlin, eric@arcetri.astro.it
Andrea Verdini, verdini@arcetri.astro.it

MODIFICATION HISTORY:
10 Oct 2005 EB/AV Created
07 Nov 2005 EB Adapted for shell-atm: rho...
27 Mar 2006 EB Added aexp, massic and volumic keywords
```

3.3.7 sa_data_merge

```
NAME:
sa_data_merge

PURPOSE:
Merge data output by successive simulations of the shell-atm model

CALLING SEQUENCE:
sa_data_merge, dirs_in, dir_out, atm=atm

INPUTS:
dirs_in: list of directories to merge
dir_out: directory where to put merged data

OPTIONAL INPUTS:
atm: if 1 (default), is for shell-atm, if 0, is for shell-loop

SIDE EFFECTS:
Creates and fills dir_out

RESTRICTIONS:
. dirs_in should contain successive parts of the same simulation.
```

```
. Unix-only (see spawn)

LANGUAGE:
IDL

PROJECT:
shell IAS

TODO:
Merge also out_simstate_nnn and time2i (need to remap time2i and nnn?)

AUTHORS:
Eric Buchlin, http://eric.buchlin.org/

MODIFICATION HISTORY:
21 Apr 2004 EB Creation
07 Nov 2005 EB Adapted to shell-atm
```

3.3.8 sa_params_write

```
NAME:
sa_params_write

PURPOSE:
Write parameters for a shell_loop simulation in a fake simstate.dat
file that can be used for data analysis or to start a simulation

CALLING SEQUENCE:
sa_params_write, p, "simstate.dat", shells=shells, time=time,
forc=forc, atm=atm

INPUTS:
p (of type sa_params): parameters
"simstate.dat" : simulation state file

OPTIONAL INPUT:
shells: values of shells (real array, with real and imaginary parts
of the complex values of the shells). Default is an array of zeroes.
time: the time of the simulation. Deprecated; use p.time instead
(but time overrides p.time)
forc: coefficients of forcing
(complex, dimensions (2[coefficients], nkforce, 2[boundaries]))
atm: if 1 (default), is for shell-atm, if 0, is for shell-loop

OUTPUT:
p : sa_params structure containing the parameters

OPTIONAL OUTPUTS:
shells: last shell values (in all planes)

RESTRICTIONS:
Assumes that nkforce=3 if forc not provided

LANGUAGE:
IDL
```

```
PROJECT:
shell-loop, shell-atm

AUTHORS:
Eric Buchlin, eric@arcetri.astro.it

MODIFICATION HISTORY:
22 Apr 2004 EB Creation
07 Jun 2005 EB Support for p.time. Format for double precision output
07 Nov 2005 EB Adapted for shell-atm
17 Feb 2006 EB Added input of forcing coefficients
```

3.3.9 sa_plot_cutk

```
NAME:
sa_plot_cutk

PURPOSE:
Read necessary information and plot cut of (z,k,t) data cube of
field (square amplitude --- massic energy) for a given k

CALLING SEQUENCE:
sa_plot_cutk, k, sf, /uu, /bb, /zp, /zm, time=time, /log,
/nointerpol, range=range,
gamma=gamma, atm=atm, /energy

INPUTS:
k: index of kperp used to plot the cut

OPTIONAL INPUTS:
sf: file simstate.dat (default) to get paramaters (data will be
taken from other out_simstate_* files)

KEYWORD PARAMETERS:
One of /uu, /bb, /zm and /zp is needed:
/uu: plot only velocity spectra
/bb: plot only magnetic field spectra
/zm: plot only zm spectra
/zp: plot only zp spectra
time: arrays of indices of times (from out_time2i) to plot
/log: shows intensities in logarithmic scale
/nointerpol: do not interpolate coordinates in plot_image (this may
cause axes values to be false)
range: the range of the color scale will be reduced (but not extended)
to fit this range
gamma: gamma factor for displaying image (in the case of a cut).
atm: if 1 (default), is for shell-atm, if 0, is for shell-loop
/energy: if set, read density in file and plot volumic energy
instead of massic energy

OUTPUT:

KEYWORD OUTPUT:

SIDE EFFECTS:
Show plot
```

```
LANGUAGE:
IDL

PROJECT:
shell-loop, shell-atm

AUTHORS:
Eric Buchlin, eric@arcetri.astro.it
Andrea Verdini, verdini@arcetri.astro.it

MODIFICATION HISTORY:
10 Oct 2005 EB/AV Forked from sa_plot_spectra
07 Nov 2005 EB Adapted for shell-atm
27 Mar 2006 EB Take aexp into account
```

3.3.10 sa_plot_cutt

```
NAME:
sa_plot_cutt

PURPOSE:
Read necessary information and plot cut of (z,k,t) data cube of
field (square amplitude --- massic energy) for a given t

CALLING SEQUENCE:
sa_plot_cutt, it2, sf, /uu, /bb, /zp, /zm, /log,
/nointerpol, range=range,
gamma=gamma, atm=atm, /energy

INPUTS:
it2: index of t (in out_time2i) used to plot the cut

OPTIONAL INPUTS:
sf: file simstate.dat (default) to get paramaters (data will be
taken from other out_simstate_* files)

KEYWORD PARAMETERS:
One of /uu, /bb, /zm and /zp is needed:
/uu: plot only velocity spectra
/bb: plot only magnetic field spectra
/zm: plot only zm spectra
/zp: plot only zp spectra
/log: shows intensities in logarithmic scale
/nointerpol: do not interpolate coordinates in plot_image (this may
cause axes values to be false)
range: range of the color scale
gamma: gamma factor for displaying image (in the case of a cut).
atm: if 1 (default), is for shell-atm, if 0, is for shell-loop
/energy: if set, read density in file and plot volumic energy
instead of massic energy
compexp: compensate values with k^compexp

OUTPUT:

KEYWORD OUTPUT:
```

```
SIDE EFFECTS:
Show plot

LANGUAGE:
IDL

PROJECT:
shell-loop, shell-atm

AUTHORS:
Eric Buchlin, eric@arcetri.astro.it
Andrea Verdini, verdini@arcetri.astro.it

MODIFICATION HISTORY:
10 Oct 2005 EB/AV Forked from sa_plot_spectra
07 Nov 2005 EB Adapted for shell-atm
27 Mar 2006 EB Take aexp into account
```

3.3.11 sa_plot_cutz

```
NAME:
sa_plot_cutz

PURPOSE:
Read necessary information and plot cut of (z,k,t) data cube of
field (square amplitude --- massic energy) for a given z

CALLING SEQUENCE:
sa_plot_cutz, z, sf, /uu, /bb, /zp, /zm, time=time, /log,
/nointerpol, range=range,
gamma=gamma, atm=atm, /energy

INPUTS:
z: index of z used to plot the cut

OPTIONAL INPUTS:
sf: file simstate.dat (default) to get paramaters (data will be
taken from other out_simstate_* files)

KEYWORD PARAMETERS:
One of /uu, /bb, /zm and /zp is needed:
/uu: plot only velocity spectra
/bb: plot only magnetic field spectra
/zm: plot only zm spectra
/zp: plot only zp spectra
time: arrays of indices of times (from out_time2i) to plot
/log: shows intensities in logarithmic scale
/nointerpol: do not interpolate coordinates in plot_image (this may
cause axes values to be false)
range: the range of the color scale will be reduced (but not extended)
to fit this range
gamma: gamma factor for displaying image (in the case of a cut).
atm: if 1 (default), is for shell-atm, if 0, is for shell-loop
/energy: if set, read density in file and plot volumic energy
instead of massic energy
```

```
OUTPUT:

KEYWORD OUTPUT:

SIDE EFFECTS:
Show plot

LANGUAGE:
IDL

PROJECT:
shell-loop, shell-atm

AUTHORS:
Eric Buchlin, eric@arcetri.astro.it
Andrea Verdini, verdini@arcetri.astro.it

MODIFICATION HISTORY:
10 Oct 2005 EB/AV Forked from sa_plot_spectra
07 Nov 2005 EB Adapted for shell-atm
27 Mar 2006 EB Take aexp into account
```

3.3.12 sa_plot_de

```
NAME:
sa_plot_de

PURPOSE:
Read necessary information and plot energy dissipation time series

CALLING SEQUENCE:
sa_plot_de, sf, /uu, /bb, /ub, /tot, /xlog, /ylog,
xrange=xrange, yrange=yrange, minmax=minmax, ytitle=ytitle, atm=atm

OPTIONAL INPUTS:
sf: file simstate.dat (default)

KEYWORD PARAMETERS:
One of /uu, /bb and /ub is needed:
/uu: plot only velocity spectra
/bb: plot only magnetic field spectra
/ub: plot both, on same plot
/tot: plot total energy (sum of u and b)
/xlog: logarithmic x-axis
/ylog: logarithmic y-axis
xrange: range of x-axis
yrange: range of y-axis
minmax: if set, the plot contains minmax couples of
points, with the min and max of the time series over some
interval around the point (saves memory or postscript file size)
ytitle: title for y-axis
atm: if 1 (default), is for shell-atm, if 0, is for shell-loop

OUTPUT:
```

```
KEYWORD OUTPUT:

SIDE EFFECTS:
Show plot

LANGUAGE:
IDL

PROJECT:
shell-loop, shell-atm

AUTHORS:
Eric Buchlin, eric@arcetri.astro.it

MODIFICATION HISTORY:
20 Jan 2004 EB Forked from sa_plot_energy, added xrange and yrange
12 May 2004 EB Added minmax keyword
01 Sep 2004 EB Added ytitle keyword
07 Nov 2005 EB Adapted for shell-atm
```

3.3.13 sa_plot_dt

```
NAME:
sa_plot_dt

PURPOSE:
Read necessary information and plot time step and time scales, as a
function of time, or as a function of perpendicular wavenumber k
for a given time, or as a function of both k and position z

CALLING SEQUENCE:
sa_plot_dt, sf, xrange=xrange, yrange=yrange, /taunl, /k, /zk,
minmax=minmax, /xlog, hyper=hyper, atm=atm, bisk=bisk, title=title

OPTIONAL INPUTS:
sf: file simstate.dat (default)

KEYWORD PARAMETERS:
xrange: range of x-axis
yrange: range of y-axis
/taunl: plot non-linear time scale (if out_taunl exists) instead of
time step. This is now the default: please use taunl=0 if you want
to plot the time step.
/k: plot time scales as a function of k (at the time corresponding
to sf) instead of time series of time scales
/zk: 2D plot of time scales as a function of k and z (at the time
corresponding to sf) instead of time series of time scales
xlog: plot time as a logarithmic axis
minmax: if set, the plot contains minmax couples of
points, with the min and max of the time series over some
interval around the point (saves memory or postscript file size)
/nocaption: don't plot caption for /k (which interferes with !p.multi)
hyper: hyperviscosity exponent (default for viscosity: 2)
atm: if 1 (default), is for shell-atm, if 0, is for shell-loop
bisk: value of the Biskamp 1994 "B_0" term alpha coefficient. Then
plot also the corresponding time scale (only possible with /k)
```

```
title: plot title

OUTPUT:

KEYWORD OUTPUT:

SIDE EFFECTS:
Show plot

BUGS:
When p.nt < 0 (end of simulation determined by time instead of
number of timesteps), the number of timesteps is not available
and the timestep length cannot be computed: we assume that data is
output at every timestep... Fix: need to add this information in
what the simulation outputs to simstate.dat

With neither /k or /zk, the stratification of the atmosphere is not
taken into account

/zk is experimental

LANGUAGE:
IDL

PROJECT:
shell IAS

AUTHORS:
Eric Buchlin

MODIFICATION HISTORY:
08 Nov 2003 EB Creation
15 Dec 2003 EB Added yrange keyword
09 Aug 2004 EB Added taunl keyword
11 Aug 2004 EB Added minmax keyword
16 Aug 2004 EB Added xlog keyword
07 Nov 2005 EB Added xrange keyword and adapted for shell-atm
15 Nov 2005 EB /taunl is now the default
16 Nov 2005 EB/AV Added /k keyword (from AV's sa_plot_taunl)
29 Mar 2006 EB Added /zk keyword
02 Nov 2006 EB Added /nocaption keyword
17 Dec 2012 EB Added bisk parameter
28 Apr 2014 EB Added hyper and title keywords
```

3.3.14 sa_plot_energy

```
NAME:
sa_plot_energy

PURPOSE:
Read necessary information and plot energy in box

CALLING SEQUENCE:
sa_plot_energy, sf, /uu, /bb, /ub, /tot, /xlog, /ylog,
xrange=xrange, yrange=yrange, minmax=minmax, atm=atm
```

```
OPTIONAL INPUTS:
sf: file simstate.dat (default)

KEYWORD PARAMETERS:
One of /uu, /bb and /ub is needed:
/uu: plot only velocity spectra
/bb: plot only magnetic field spectra
/ub: plot both, on same plot
/tot: plot total energy (sum of u and b)
/xlog: logarithmic x-axis
/ylog: logarithmic y-axis
xrange: range on x-axis
yrange: range on y-axis
minmax: if set, the plot contains minmax couples of
points, with the min and max of the time series over some
interval around the point (saves memory or postscript file size)
atm: if 1 (default), is for shell-atm, if 0, is for shell-loop

OUTPUT:

KEYWORD OUTPUT:

SIDE EFFECTS:
Show plot

LANGUAGE:
IDL

PROJECT:
shell IAS

AUTHORS:
EB: Eric Buchlin

MODIFICATION HISTORY:
11 Nov 2003 EB Creation
17 Dec 2003 EB Added /tot and /xlog keywords
21 Jan 2004 EB Added xrange and yrange keywords
05 Aug 2004 EB Added minmax keyword
07 Nov 2005 EB Adapted for shell-atm
```

3.3.15 sa_plot_enflux

```
NAME:
sa_plot_enflux

PURPOSE:
Read necessary information and plot energy flux in longitudinal
dimension as a function of time and position

CALLING SEQUENCE:
sa_plot_enflux, sf, time=time, /log,
/nointerpol, range=range, k=k, /ch,
xrange=xrange, yrange=yrange, gamma=gamma, atm=atm,
```

```
INPUTS:

OPTIONAL INPUTS:
sf: file simstate.dat (default) to get paramaters (data will be
taken from other out_simstate_* files)

KEYWORD PARAMETERS:
time: arrays of indices of times (from out_time2i) to plot
/log: shows intensities in logarithmic scale
/nointerpol: do not interpolate coordinates in plot_image (this may
cause axes values to be false)
range: the range of the color scale will be reduced (but not extended)
to fit this range
k: if specified, plot energy flux corresponding only to these
perpendicular wavenumber
xrange: range for x axis (not for spectrum cuts)
yrange: range for y axis (not for spectrum cuts)
gamma: gamma factor for displaying image (in the case of a cut).
atm: if 1 (default), is for shell-atm, if 0, is for shell-loop

OUTPUT:

KEYWORD OUTPUT:

SIDE EFFECTS:
Show plot

LANGUAGE:
IDL

PROJECT:
shell-loop, shell-atm

AUTHORS:
Eric Buchlin, eric@arcetri.astro.it
Andrea Verdini, verdini@arcetri.astro.it

MODIFICATION HISTORY:
11 Oct 2005 EB/AV Forked from sa_plot_cutk
07 Nov 2005 EB Adapted for shell-atm
19 Nov 2005 EB Adapted for case when out_simstate_nnn doesn't
contain all planes
27 Mar 2006 EB aexp is now taken into account by sa_shells_to_field
```

3.3.16 sa_plot_heatfunc

```
NAME:
sa_plot_heatfunc

PURPOSE:
Read necessary information and plot heating function (dissipation
power per unit length)

CALLING SEQUENCE:
sa_plot_heatfunc, sf, /log, /nointerpol, range=range, time=time
xrange=xrange, yrange=yrange, cutt=cutt, cutz=cutz, /ddd,
```

```
xtitle=xtitle, ytitle=ytitle, title=title,
gamma=gamma, profile=profile, atm=atm

OPTIONAL INPUTS:
sf: file simstate.dat (default)

KEYWORD PARAMETERS:
/log: if a 2D cut is plotted, shows intensities in logarithmic
scale
/nointerpol: do not interpolate coordinates in plot_image (this may
cause axes values to be false)
time: plot only for these times (expressed as an index of the
time dimension of the spectra data cube; is between 0 and
p.nrecsp - 1). time can be an array.
cutt: plot only cut(s) of the 2D heating function for the given time(s)
cutz: plot only cut(s) of the 2D heating function for the given z
ddd: plot surface plot instead of 2D image
range: for the cuts:
if range is a 2-element array, the range of the color
scale will be reduced (but not extended) to fit this
range
if it is a scalar different from 0, the range of the
color scale will match the range of values of the whole
data cube
if it is 0 or not set, the contrast will be maximized for
each plot
gamma: gamma for displaying image
xrange, yrange: ranges for x and y axes (only for profile)
xtitle, ytitle, title: titles for axes and for plot
profile: plot profile of heating along the model
=1 : read out_profheat
=2 : compute heating profile from out_simstate_nnnnnnnnnn
/volumic: plot dissipation power per unit volume instead of unit
length (no effect if atm is 0)
atm: if 1 (default), is for shell-atm, if 0, is for shell-loop

SIDE EFFECTS:
Show plot

LANGUAGE:
IDL

PROJECT:
shell IAS

AUTHORS:
Eric Buchlin, eric@arcetri.astro.it

MODIFICATION HISTORY:
22 Apr 2004 EB Creation
16 Jul 2004 EB Added gamma keyword
07 Nov 2005 EB Adapted for shell-atm
10 Nov 2005 EB Added ddd, cutt and cutz keywords
19 Nov 2005 EB Adapted for case when out_simstate_nnn doesn't
contain all planes
12 Dec 2005 EB Added xtitle, ytitle, and title keywords
```

```
27 Mar 2006 EB Added profile and volumic keywords
15 May 2006 EB Added profile=1 or 2 option
```

3.3.17 sa_plot_specav

```
NAME:
sa_plot_specav

PURPOSE:
Read necessary information and plot averaged energy in each shell
(or spectra) (summed over z)

CALLING SEQUENCE:
sa_plot_specav, sf, /uu, /bb, /ub, /bu, /tot, time=time, , shift=shift
/heating, /image, /nointerpol, xrange=xrange, yrange=yrange,
atm=atm, taverage=taverage, /realspec, colors=colors,
/diffcons, /diffav, compexp=compexp

OPTIONAL INPUTS:
sf: file simstate.dat (default)

KEYWORD PARAMETERS:
One of /uu, /bb, /ub and /tot is needed:
/uu: plot only velocity spectra
/bb: plot only magnetic field spectra
/ub: plot both, on separate plots (!p.multi must be set before
calling this procedure if necessary)
/bu: plot both, on same plot (u in black or white, b in blue
time: plot spectra only for these times (expressed as an index of the
time dimension of the spectra data cube; is between 0 and
p.nrecsp - 1). time can be an array. The spectra are taken _at the
middle of the loop_.
shift: shift (in orders of magnitude) on the y axis to "pile up"
the spectra
/heating: multiply by nu or eta k^2, so as to have heating instead
of spectrum
/image: produce image with axes (k,t). In this case, the color range
is yrange
/nointerpol: no interpolation for image plot
xrange: range for x axis
yrange: range for y axis
atm: if 1 (default), is for shell-atm, if 0, is for shell-loop
taverage: plot a number taverage of partial averages over given
times instead of all spectra. 1: plot global average, 2: plot
average of first half of provided times, plus average of second half...
/realspec: plot real 1D spectrum (instead of energy as a function
of shell)
/slope: plot slope of spectrum instead of the spectrum itself
colors: colors of curves at different times (24-bit true color values)
/diffcons: plot differences between consecutive spectra (in log10
space)
/diffav: plot differences with average spectrum (in log10 space)
compexp: compensate power-law by multiplying spectrum by k^compexp
compshift: compensate constant by multiplying spectrum by compshift

OUTPUT:
```

```
KEYWORD OUTPUT:

SIDE EFFECTS:
Show plot

LANGUAGE:
IDL

PROJECT:
shell-atm

AUTHORS:
EB Eric Buchlin

MODIFICATION HISTORY:
07 Nov 2005 EB Forked from sl_plot_spectra
15 Nov 2005 EB Added shift keyword
19 Nov 2005 EB Added /image and /heating keywords
23 Jan 2006 EB Added /tot keyword
24 Jan 2006 EB Added /taverage and /realspec keywords
29 Jan 2006 EB Added /slope keyword
24 May 2006 EB Added colors keyword, extended meaning of taverage
29 May 2006 EB Added /diffcons and /diffav keywords
02 Apr 2007 EB Added compexp and compshift keywords
```

3.3.18 sa_plot_spectrum2d

```
NAME:
sa_plot_spectrum2d

PURPOSE:
Read necessary information and plot 2D spectra

CALLING SEQUENCE:
sa_plot_spectrum2d, sf, /uu, /bb, /zm, /zp, /tot, /nointerpol, /isotropic,
range=range, xrange=xrange, yrange=yrange, ks=ks, slope=slope,
/contour, /thcont, time=time, /noplot, spec=spec, kperp=kperp,
kpar=kpar, atm=atm

OPTIONAL INPUTS:
sf: file simstate.dat (default)

KEYWORD PARAMETERS:
One of /uu, /bb, /zm, /zp or /tot needed:
/uu: plot velocity field spectrum
/bb: plot magnetic field spectrum
/zp: plot z+ Elsässer field spectrum
/zm: plot z- Elsässer field spectrum
/tot: plot total field spectrum

/nointerpol: do not interpolate coordinates in plot_image (this may
cause axes values to be false)
range: maximum data range for color scale
xrange: range for x axis
yrange: range for y axis
```

```
/isotropic: isotropic plot
/contour: overplot contours
/thcont: overplot thererical contour
/han: hanning window
/ktheta: theta-logk plot
/polar: polar plot
/noplot: don't do plot (useful when called for spec output)
time: array of times (of indices of non-integrated output times) on
which to average (otherwise use sf)

atm: if 1 (default), is for shell-atm, if 0, is for shell-loop

OPTIONAL OUTPUTS:
spec: 2D spectrum
kperp: array of perpendicular wavenumbers
kpar : array of parallel wavenumbers

SIDE EFFECTS:
Show plot

RESTRICTIONS:
Assume that all out_simstate_nnn have the same nmin, nmax and nz

LANGUAGE:
IDL

PROJECT:
shell-loop, shell-atm

AUTHORS:
Eric Buchlin, eric.buchlin@ias.u-psud.fr
Andrea Verdini, verdini@arcetri.astro.it

MODIFICATION HISTORY:
13 Feb 2004 EB Forked from sa_plot_spectra
07 Nov 2005 EB Adapted for shell-atm
09 Dec 2005 EB Take into account precise surface of shells (but
still missing other constant factors)
06 Feb 2005 EB time, /thcont, /noplot, spec, kperp and kpar keywords
```

3.3.19 sa_plot_spectrumpp

```
NAME:
sa_plot_spectrumpp

PURPOSE:
Read necessary information and plot spectra in perpendicular and
parallel direction

CALLING SEQUENCE:
sa_plot_spectrumpp, sf, /uu, /bb, /zm, /zp, /tot
xrange=xrange, yrange=yrange, atm=atm, cutk=cutk

OPTIONAL INPUTS:
sf: file simstate.dat (default)
```

```
KEYWORD PARAMETERS:
One of /uu, /bb, /zm, /zp or /tot needed:
/uu: plot velocity field spectrum
/bb: plot magnetic field spectrum
/zp: plot z+ Elsässer field spectrum
/zm: plot z- Elsässer field spectrum
/tot: plot total field spectrum

cutk: plot cuts in 2D spectrum for parallel and perpendicular
wavenumbers cutk

xrange: range for x axis
yrange: range for y axis
time: array of times (of indices of non-integrated output times) on
which to average (otherwise use sf)
atm: if 1 (default), is for shell-atm, if 0, is for shell-loop

SIDE EFFECTS:
Show plot

LANGUAGE:
IDL

PROJECT:
shell IAS

AUTHORS:
Eric Buchlin, eric@arcetri.astro.it

MODIFICATION HISTORY:
01 Nov 2004 EB Forked from sa_plot_spectrum2d
07 Nov 2005 EB Adapted for shell-atm
09 Dec 2005 EB Take into account precise surface of shells (but
still missing other constant factors)
09 Dec 2005 EB Added cutk keyword
06 Feb 2006 EB Added time keyword. Replaced computation of 2D
spectrum by call to sa_plot_spectrum2d
```

3.3.20 sa_shells_to_fields

```
NAME:
sa_shells_to_fields

PURPOSE:
Transforms shell values array into fields and energies. Default is
to get energies per unit length, but energies per unit mass or per
unit volume can also be obtained.

CALLING SEQUENCE:
s = sa_shells_to_fields (shells, /uu, /bb, /zp, /zm, /energy,
rho=rho, aexp=aexp, k0=k0, /massic,
/volumic, atm = atm)

INPUTS:
shells: array of complex shell values,
3 dimensions:
```

```
- first dimensions of the array can be anything
- second dimension: must correspond to the position on
the loop if computing the energies
- last dimension: 0 is zp, 1 is zm

KEYWORD PARAMETERS:
/uu, /bb, /zp, /zm (one of these is needed): field to compute
/energy: compute lineic energy in field instead of returning the
amplitudes of the field
rho: mass density of plasma
aexp: over-expansion factor (in area): the width of the loop at a
given position is the width deduced from k0, multiplied by square
root of aexp at this position
k0: k0 wavenumber of simulation
/massic:  get energy per unit mass instead of energy per unit length
/volumic: get energy per unit volume instead of energy per unit length
atm: if 1 (default), is for shell-atm, if 0, is for shell-loop

OUTPUTS:
Field or energy of field

SIDE EFFECTS:
None

PROJECT:
shell-loop, shell-atm

AUTHORS:
Eric Buchlin, eric@arcetri.astro.it
Andrea Verdini, verdini@arcetri.astro.it

MODIFICATION HISTORY:
10 Oct 2005 EB/AV Created
07 Nov 2005 EB Adapted for shell-atm: rho...
27 Mar 2006 EB Added aexp, massic and volumic keywords
```

3.3.21 sa_specmov

```
NAME:
sa_specmov

PURPOSE:
Create movie of average spectra, read in out_specavu and
out_specavb
Outputs PNG files in current directory, that can be converted to a
movie: with mplayer/mencoder, use:
mencoder "mf://*.png" -fps 25 -ovc lavc -lavcopts
vcodec=mpeg4:vbitrate=600000 -ffourcc DX50 -o specmov.avi

CALLING SEQUENCE:
sa_specmov, sf, xrange=xrange, yrange=yrange, time=time, /ts, /usemencoder,
atm=atm

OPTIONAL INPUTS:
sf: file simstate.dat (default)
```

```
KEYWORD PARAMETERS:
xrange: range for x axis
yrange: range for y axis
time: indices of time output to use (default: all (0 to p.nrecsp-1))
/ts: plot time series of energy and dissipation in separate panels
/usemencoder: try to use mencoder to encode AVI movie
atm: if 1 (default), is for shell-atm, if 0, is for shell-loop

OUTPUTS:

SIDE EFFECTS:
Show plots of spectra
Create PNG files (need to be deleted manually in the end)
Create AVI movie (if /mencoder is used and mencoder available)

LANGUAGE:
IDL

PROJECT:
shell-atm

AUTHORS:
Eric Buchlin, eric@arcetri.astro.it

MODIFICATION HISTORY:
14 Nov 2005 EB Creation
```

Other routines are provided but they are not intended to be used directly by the end user. Please see the documentation included in their source code for further details. Some of these routines may depend on other routines from EB's personal library: please contact him to get the dependent files.

4 Programming the code

4.1 Types and variables

Variables shared between modules and subroutines, as well as user-defined types, are contained in `src/types.f90`. Other variables are passed to routines as arguments. There shouldn't be any global variables.

4.1.1 User-defined types

Summary of the types:

- `params`: parameters of the simulation, contains `nmin`, `nmax`, `tnz`, `nz`, `nz0`, `k0`, `lambda`, `nu`, `eta`, `eps`, `epsm`, `b0`, `rho0`, `dz`, `nt`, `tmax`, `delta_t`, `continue`, `initamp`, `initslope`, `forcamp`, `tsar`, `h`, `pexp` (of type `paramexp`), `forcp`, `mpi` (of type `mpiparams`), `fc` (of type `forcecoeffs`)
- `forcecoeffs`: last random coefficients of forcing, contains `a1`, `b1`, `a2`, `b2`, all of dimension `nkforce`.
- `mpiparams`: parameters of the parallel run, contains `me`, `np`, `left`, `right`, `root`.
- `paramexp`: parameters for the expansion function, contains `x_0`, `sigma`, `fmax`, `x1n`
- `saveflags`: frequency of output and flags, contains
  - the periods and parameters for output `period`, `periodt`, `specperiod`, `specperiodt`, `sfunskip`, `sfunnb`

  - the flags `time`, `taunl`, `enmode02`, `enmode08`, `enmode14`, `enmode20`, `entotu`, `entotb`, `dissu`, `dissb`, `forcp`, `crosshel`, `maghel`, `specu`, `specb`, `sfunu`, `sfunb`, `rawzp`, `rawzm`, `rad`, `profva`, `profvad`, `profrho`, `profaexp`

- `timescales`: time scales of the model as computed by the last call to `check_time_scales`, contains `taunl`, `taual`, `taunu`, `taucr`.
- `saverecords`: number of records saved for each type of output, contains `energy`, `sfun`, `spec`.
- `units`: values in CGS units of the units of the model, contains `length`, `time`, `mass`.

The details of these types and the description of their members can be found in `src/types.f90`.

4.1.2 Shared variables

These variables are shared through the use of `use types` in each module:

- `dfloat`: precision (Fortran `kind`) for floats with at least 15 digits and up to 10^{300} (should correspond to double precision)
- `nkforce`: number of modes on which forcing is done.
- `ierr`, `aerr`: error codes for parallel routines calls and for allocation of arrays

4.1.3 Other variables

These variables are local to each module or procedure, but they usually have a common meaning all over the code:

- `zp` (`p%nk`, `p%nz`): Elsässer field $Z^+ = u + b$ of the shell-model, propagating towards growing z (this supposes that $\vec{B}_0 \cdot \vec{e}_z < 0$). `zm` is the other Elsässer field $Z^- = u - b$, propagating in the other direction.
- `k` (`p%nk`): array of values of the wavenumber corresponding to each shell
- `ti`: timestep number, starting at 1
- `t`: time, starting at 0 or at the last time attained in the previous run if we continue from this run
- `p`, of type `params`
- `tscales`, of type `timescales`
- `outflags`, of type `saveflags`
- `outrecords`, of type `saverecords`
- `p`, of type `params`

The dimensions are given for each processor. They start at `p%nmin` for a dimension `p%nk` and at 0 otherwise. `p%nk` is defined as `p%nmax` − `p%nmin` + 1 in this manual for convenience but is not a variable of the code.

4.2 Modules, routines and functions

4.2.1 Main program: `shell-atm`

The main program contains the calls to the routines for initialization and allocation of variables, the main loop, and calls to routines for finishing the simulations.

```
MHD GOY shell-model-based coronal stratified atmosphere model
Constrained geometry, (static stratification, no wind)
Each plane cross-section is a (pseudo-2D) shell-model
(see Giuliani & Carbone 1998)
Parameters:
read in params.txt
Output and output files:
see read_output_options
Forcing: on modes 2-4 (see propagate_and_force) of photospheric planes
Structure:
Initialize:
read_parameters
read_output_options
get_wavenumbers
init_shells or read_simstate
output_open_files
Main loop:
|  Check if need to adapt time step for output
|  integrate_step
|  |  integrate_step_nonlin
|  |     derivs2d
|  |  dissipate_nuperp
|  |  propagate_wave
|  |  forcing
|  [check for numerical explosion and quit]
|  output_data_a
|  output_data_b
|  Every 10 steps: check_time_scales
|  Every 10^7/np steps: write_simstate
(repeat)
Quit:
output_close_files
write_simstate
Authors:
AV   Andrea Verdini
EB   Eric Buchlin
Modifications history:
28 Nov 02: EB forked from shell_mhd
10 Dec 02: EB Alfven wave propagation
06 Nov 03: EB 3rd order space Alfven wave propagation and integration
in Runge-Kutta time integration
11 Dec 03: EB Added Lax-Wendroff Alfven wave propagation
no 2D k-array anymore
17 Jan 04: EB Corrected bug in RK3 scheme implementation
Photospheric forcing as boundary conditions
20 May 04: EB Alfven wave propagation included in forcing instead of
shell-models
08 Jul 04: EB Added Beam-Warming numerical scheme
10 Aug 04: EB Output of TauNL. Stop criterium on time (tmax).
Implicit dissipation in planes
14 Mar 05: EB/AV Parallel code (forked from shell-loop.f90)
14 Jul 05: AV/EB: merged shell-loop and stat-atm_sfer
07 Aug 05: EB Save random coefficients of forcing in simstate.dat
tstar is now a parameter read in params.txt
19 Aug 05: AV Completed module atmosphere (unif, cart, spher)
Added module for flux tube expansion, fexpand (none, MJ)
```

```
Added type p%pexp and parameters file paramexp.txt
19 Aug 05: AV Added left and right for the call to sendrecv in
propagate_and_force
22 Aug 05: AV Added normalization for Va, Va_m, Vad
- use CsN in the atmosphere module
- initial and forcing amplitude input as normalized
22 Aug 05: AV To check numerical scheme added some TRUE/FALSE option
to be changed before compiling
- added simple starting amplitude at the base
(in accordance with the forcing)
22 Aug 05: AV Corrected bug in isend irecv in 'propagate_and_force'
- missing wait fot ozm
Program shell_atm
use mpi
use types
use atmosphere
use nonlin
use initialize
use diagnostics
use iodata
use ioparams
use iosimstate

Routines for equation integration

Perform one step of the shell-model time integration
MPI: OK (inherited from integrate_step_nonlin, dissipate_nuperp,
propagate_wave and forcing)
subroutine integrate_step (zp, zm, k, p, t)
use boundary
use propagation

Implicit scheme for dissipation in planes
Assumes a Prandtl number = 1: p%eta = p%nu
MPI: OK (no need)
subroutine dissipate_nuperp (zp, zm, k, p)
```

4.2.2 Module `atmosphere`

This module provides variables and routines to describe the atmosphere, with several alternatives. Each alternative should provide the variables:

- `module_atmosphere`: identification of the module alternative
- `va`: Alfvén velocity profile (1D array of size `nz`)
- `va_m`: Alfvén velocity profile at mid-point (1D array of size `nz`)
- `vad`: Alfvén velocity gradient profile (1D array of size `nz`)
- `rho`: mass density profile (1D array of size `nz`)
- `r`: position of grid points (used only to build atmosphere; grid points are still assumed to be separated by δz, and r is not needed elsewhere in the code) (1D array of size `nz`)
- `aexp`: profile of overexpansion factor (1D array of size `nz`)
- `kaexp`: inverse square root of expansion factor (1D array of size `nz`)

and the routines:

- `atmosphere_allocate`: allocate variables for the atmosphere
- `atmosphere_deallocate`: deallocate variables for the atmosphere
- `atmosphere_print_parameters`: display parameters of atmosphere

These may be used by `grid_assign`:

- `r_det`: position as a function of index in the z direction
- `va_det`: Alfvén velocity as a function of position
- `vad_det`: Alfvén velocity gradient as a function of position
- `b_det`: magnetic field as a function of position
- `rho_det`: mass density as a function of position
- `aexp_det`: over-expansion factor as a function of position

The current alternatives are:

`src/atmosphere-cart.f90` for a cartesian atmosphere.

```
Module atmosphere-cart
Variables and routines for a cartesian atmosphere
Authors:
AV   Andrea Verdini
EB   Eric Buchlin
Modifications history:
19 Aug 05: AV Created
module atmosphere
use types
use fexpand

Allocate variables for the atmosphere
subroutine atmosphere_allocate(p)

Deallocate variables for the atmosphere
subroutine atmosphere_deallocate

Assign values for Alfvén speed, Alfvén speed gradient and
mass density to the grid points.
These values are computed by the xxx_det routines
(different alternatives according to geometry)
MPI: done on each processor separately (but need to check if OK)
subroutine grid_assign (p)

Determination of position/altitude as a function of index of plane
function r_det (i, p)

Determination of Alfvén velocity as a function of altitude
function va_det (r, p)

Determination of Alfvén velocity gradient as a function of altitude
function vad_det (r, p)

Determination of magnetic field as a function of altitude
```

```
function b_det (r, p)

Determination of density as a function of altitude
function rho_det (r, p)

Determination of flux tube expansion as a function of altitude
function aexp_det (r, p)

Determination of logaritmic density gradient as a function of altitude
function rhodl_det (r, p)

Determination of flux tube expansion logarithmic derivative A'/A
as a function of altitude: A = r^2*f
function aexpdl_det(r, p)

Determination sound speed for normalisation (if needed)
function cs_det (p)

Print parameters specific to atmosphere
subroutine atmosphere_print_parameters(p)
```

src/atmosphere-dmit.f90 for an atmosphere according to Dmitruk *et al.* (2001).

```
Module atmosphere-dmit
Variables and routines for a Dmitruk-like atmosphere
Dmitruk 2001 (ADS: 2001ApJ...548..482D)
Authors:
AV   Andrea Verdini
EB   Eric Buchlin
Modifications history:
19 Aug 05: AV: created
module atmosphere
use types

Allocate variables for the atmosphere
subroutine atmosphere_allocate(p)

Deallocate variables for the atmosphere
subroutine atmosphere_deallocate

Assign values for Alfvén speed, Alfvén speed gradient and
mass density to the grid points.
These values are computed by the xxx_det routines
(different alternatives according to geometry)
MPI: done on each processor separately (but need to check if OK)
subroutine grid_assign (p)

Determination of position/altitude as a function of index of plane
function r_det (i, p)

Determination of Alfvén velocity as a function of altitude
function va_det (r, p)

Determination of Alfvén velocity gradient as a function of altitude
function vad_det (r, p)
```

```
Determination of magnetic field as a function of altitude
function b_det (r, p)

Determination of density as a function of altitude
function rho_det (r, p)

Determination of flux tube expansion as a function of altitude: A = r^2*f
function aexp_det (r, p)

Determination sound speed for normalisation (if needed)
function cs_det (p)
use unit

Print parameters specific to atmosphere
subroutine atmosphere_print_parameters(p)
```

src/atmosphere-spher.f90 for a spherical atmosphere

```
Module atmosphere-spher
Variables and routines for a spherical atmosphere
Authors:
AV   Andrea Verdini
EB   Eric Buchlin
Modifications history:
15 Jul 05 AV Created
module atmosphere
use types
use fexpand
use unit

Allocate variables for the atmosphere
subroutine atmosphere_allocate(p)

Deallocate variables for the atmosphere
subroutine atmosphere_deallocate

Assign values for Alfvén speed, Alfvén speed gradient and
mass density to the grid points.
These values are computed by the xxx_det routines
(different alternatives according to geometry)
MPI: done on each processor separately (but need to check if OK)
subroutine grid_assign (p)

Determination of position/altitude as a function of index of plane
function r_det (i, p)

Determination of Alfvén velocity as a function of altitude
function va_det (r, p)

Determination of Alfvén velocity gradient as a function of altitude
function vad_det (r, p)

Determination of magnetic field as a function of altitude
function b_det (r, p)
```

```
Determination of density as a function of altitude
function rho_det (r, p)

Determination of logarithmic derivative of density
as a function of altitude
function rhodl_det (r, p)

Determination of flux tube expansion as a function of altitude: A = r^2*f
function aexp_det(r, p)
use fexpand

Determination of flux tube expansion logarithmic derivative A'/A
as a function of altitude: A = r^2*f
function aexpdl_det (r, p)
use fexpand

Determination sound speed for normalisation (if needed)
function cs_det (p)

Print parameters specific to atmosphere
subroutine atmosphere_print_parameters(p)
```

src/atmosphere-stratin.f90 for atmosphere whose Alfvén speed profile is read in `in_va` and mass density profile is read in `in_rho`. The expansion factor profile is then deduced from the conservation of the magnetic flux in the flux tube. Note that the nominal width of the flux tube (*i.e.* the point vers the expansion factor is 1) is taken at the middle of the length of the flux tube.

```
Module atmosphere-stratin
Variables and routines for a stratified cartesian atmosphere where
the profiles for density and Alfven speed are read in files
(in_rho and in_va)
Authors:
AV   Andrea Verdini
EB   Eric Buchlin
Modifications history:
22 Mar 2006: EB Created
module atmosphere
use types

Allocate variables for the atmosphere
subroutine atmosphere_allocate(p)

Deallocate variables for the atmosphere
subroutine atmosphere_deallocate

Assign values for Alfven speed, Alfven speed gradient and
mass density to the grid points.
These values are computed by the xxx_det routines
(different alternatives according to geometry)
MPI: need to check
subroutine grid_assign (p)
use mpi

Print parameters specific to atmosphere
```

```
subroutine atmosphere_print_parameters(p)
```

src/atmosphere-unif.f90 for a cartesian atmosphere with uniform profiles of longitudinal magnetic field and density

```
Module atmosphere-unif
Variables and routines for a cartesian atmosphere and uniform Va
Authors:
AV   Andrea Verdini
EB   Eric Buchlin
Modifications history:
19 Aug 05: AV: created
module atmosphere

Allocate variables for the atmosphere
subroutine atmosphere_allocate(p)

Deallocate variables for the atmosphere
subroutine atmosphere_deallocate

Assign values for Alfvén speed, Alfvén speed gradient and
mass density to the grid points.
These values are computed by the xxx_det routines
(different alternatives according to geometry)
MPI: done on each processor separately (but need to check if OK)
subroutine grid_assign (p)

Determination of position/altitude as a function of index of plane
function r_det (i, p)

Determination of Alfvén velocity as a function of altitude
function va_det (r, p)

Determination of Alfvén velocity gradient as a function of altitude
function vad_det (r, p)

Determination of magnetic field as a function of altitude
function b_det (r, p)

Determination of density as a function of altitude
function rho_det (r, p)

Determination of flux tube expansion as a function of altitude: A = r^2*f
function aexp_det(r, p)

Determination sound speed for normalisation (if needed)
function cs_det (p)

Print parameters specific to atmosphere
subroutine atmosphere_print_parameters(p)
```

4.2.3 Module boundary

This module provides variables and routines to describe the forcing and the boundary conditions of the model, with several alternatives. Each alternative should provide the variables:

- `module_boundary`: identification of the module alternative

and the routines:

- `force`: compute `zp` on first plane and `zm` on last plane to impose boundary conditions and forcing

The current alternatives are:

`src/boundary-closed` for total reflection on both boundaries (no forcing)

```
Module boundary-closed
Variables and routines for boundary conditions and forcing
For a closed box (total reflection on both boundaries)
Authors:
AV   Andrea Verdini
EB   Eric Buchlin
Modifications history:
10 Nov 05 EB Created
module boundary
use types

Boundary conditions (reflection and forcing)
Total reflection
MPI: OK (done)
subroutine force (zp, zm, k, p, t)
use mpi
```

`src/boundary-periodic` for periodic boundary conditions (then forcing is done in the volume; to be used with module `propagation-periodicfromm`)

```
Module boundary-periodic
Variables and routines for boundary conditions and forcing
For periodic boundary conditions
Velocity forcing is imposed in the volume of the box.
It is the same everywhere in the box seen by one processor,
but it differs from one processor to the next!
Authors:
AV   Andrea Verdini
EB   Eric Buchlin
Modifications history:
04 Oct 2012 EB Created (forked from boundary-photospheres)
module boundary
use types

Periodic boundary conditions: do nothing at boundaries,
but force in volume.
(periodicity will be implemented in propagation scheme)
MPI: OK (done), but check whether r_det is parallelized!
subroutine force (zp, zm, k, p, t)
```

`src/boundary-photoopen` for a photosphere at the bottom and an open boundary condition at the top (`zm` = 0)

```
Module boundary-zpopen
Variables and routines for boundary conditions and forcing
For imposed zp on one side and open boundary condition the other side
Authors:
AV   Andrea Verdini
EB   Eric Buchlin
Modifications history:
07 Nov 05 EB Created
module boundary
use types

Boundary conditions (reflection and forcing)
Imposed zp, open at top
MPI: OK (done)
subroutine force (zp, zm, k, p, t)
use mpi
use atmosphere
use diagnostics
```

`src/boundary-photosphere.f90` for a photosphere at the bottom and total reflection at the top

```
Module boundary-photosphere
Variables and routines for boundary conditions and forcing
For a photosphere on one side and total reflection on the other side
Authors:
AV   Andrea Verdini
EB   Eric Buchlin
Modifications history:
07 Nov 05 EB Created
module boundary
use types

Boundary conditions (reflection and forcing)
A velocity is imposed, which leads to partial reflection
MPI: OK (done)
subroutine force (zp, zm, k, p, t)
use mpi
use atmosphere
use diagnostics
```

`src/boundary-photospheres.f90` for a photosphere at both boundaries (like for a coronal loop)

```
Module boundary-photospheres
Variables and routines for boundary conditions and forcing
For a photosphere on both sides of the box
Authors:
AV   Andrea Verdini
EB   Eric Buchlin
Modifications history:
07 Nov 05 EB Created
module boundary
use types

Boundary conditions (reflection and forcing)
```

```
A velocity is imposed, which leads to partial reflection
MPI: OK (done)
subroutine force (zp, zm, k, p, t)
use mpi
use atmosphere
use diagnostics
```

`src/boundary-zpopen` for imposing `zp` at the bottom and an open boundary condition at the top (`zm` = 0)

```
Module boundary-zpopen
Variables and routines for boundary conditions and forcing
For imposed zp on one side and open boundary condition the other side
Authors:
AV   Andrea Verdini
EB   Eric Buchlin
Modifications history:
07 Nov 05 EB Created
module boundary
use types

Boundary conditions (reflection and forcing)
Imposed zp, open at top
MPI: OK (done)
subroutine force (zp, zm, k, p, t)
use mpi
use atmosphere
use diagnostics
```

4.2.4 Module `diagnostics`

This module, in the file `src/diagnostics.f90`, contains routines for evaluating integrated quantities of the fields, to perform conversion between Elsässer fields and velocity and magnetic field, to check for time scales, and to evaluate the simulation efficiency.

```
Module diagnostics
Computation of diagnostics from fields, integrated quantities, conversions
Authors:
AV   Andrea Verdini
EB   Eric Buchlin
Modifications history:
22 Oct 05: EB Created (moved from shell-atm)
05 Nov 05: EB Take density into account
module diagnostics
use mpi
use types
use atmosphere

Return the total energy (massic density) contained in shells
of a loop cross-section
As we don't know the density in this section or its position,
we can't take it into account right here
MPI: OK (no need)
function energy_tot1d (z, p)
```

```
Return the total energy contained in one shell on all planes
Take into acount mass density, and separation between planes
MPI: OK (done)
function energy_tot1dk (z, p)

Return the total energy contained in shells of a field (whole
box, all processors)
Take into acount mass density, and separation between planes
MPI: OK (done)
function energy_tot (z, p)

Return the total energy contained in shells of both fields of a
loop section
As we don't know the density in this section or its position,
we can't take it into account right here
MPI: OK (no need; however, when used [out of specifications, and deprecated]
to compute enmodexx, needs to be considered as not parallel)
function energy_2_tot1d (z1, z2, p)

Return the total energy contained in one shell of both fields in
all planes
Take into acount mass density, and separation between planes
(inherited from energy_tot1d)
MPI: OK (done)
function energy_2_tot1dk (z1, z2, p)

Return the total energy contained in shells of both fields
(whole box, all processors)
Take into acount mass density, and separation between planes
(inherited from energy_tot)
MPI: OK (inherited from energy_tot)
function energy_2_tot (z1, z2, p)

Return the energy corresponding to a shell value
As we don't know the density in this section or its position,
we can't take it into account right here
MPI: OK (no need)
function energy (zval)

Return the cross-helicity H_C (whole box, all processors)
Take into acount mass density, and separation between planes
(inherited from energy_tot)
MPI: OK (inherited from energy_tot)
function cross_helicity (zp, zm, p)

Return third invariant (=magnetic helicity in 3D)
of a loop cross-section
As we don't know the density in this section or its position,
we can't take it into account right here
*** in development ***
function magn_helicity1d (zp, zm, k, p)

Return third invariant (=magnetic helicity in 3D)
Take into acount mass density, and separation between planes
*** in development ***
function magn_helicity (zp, zm, k, p)
```

```
Return the total dissipation for a field u or b (whole
box, all processors)
Argument field is u or b
Take into acount mass density, and separation between planes
MPI: OK (done)
function diss_tot (z, k, p, disscoeff)

Transform (u, b) to (zp, zm) for a loop section
MPI: OK (no need)
subroutine ub_to_zpm1d (u, b, zp, zm, p)

Transform (u, b) to (zp, zm) for the planes of one processor
MPI: OK (no need)
subroutine ub_to_zpm (u, b, zp, zm, p)

Transform (zp, zm) to (u, b) for a loop section
MPI: OK (no need)
subroutine zpm_to_ub1d (zp, zm, u, b, p)

Transform (zp, zm) to (u, b) for the planes of one processor
MPI: OK (no need)
subroutine zpm_to_ub (zp, zm, u, b, p)

Compute Reynolds numbers Re and Rm (average over first
quarter of wavenumbers for a loop section)
TODO: this should be updated to a better definition of Re/Rm
MPI: OK (no need)
subroutine reynolds_numbers1d (zp, zm, k, re, rm, p)

Check whether time scales are shorter than time step
MPI: OK (done)
subroutine check_time_scales (zp, zm, k, p, ti, t, tscales)
use atmosphere
use nonlin

Initialize or print wallclock time needed for simulation
Prints information on efficiency
MPI: OK (done)
subroutine wallclock (p, ti, init)
```

4.2.5 Module `fexpand`

This module provides variables and routines to describe the over-expansion factor of the flux tube, with several alternatives. Each alternative should provide the variables:

- `module_fexpand`: identification of the module alternative

and the routines:

- `fexpansion`: expansion factor as a function of position
- `fdlexpansion`: logarithmic derivative of the expansion factor as a function of position
- `expansion_print_parameters`: display parameters specific to expansion

Note: "expansion factor" stands here for "over-expansion factor" compared to a radial expansion.

The current alternatives are:

`src/fexpand-none.f90` for no over-expansion.

```
Module fexpand-none
Routines for no expansion factor
Authors:
AV   Andrea Verdini
EB   Eric Buchlin
Modifications history:
19 Aug 05: AV Created
module fexpand
use types

Expansion factor as a function of r
function fexpansion(r, pexp)

Logarithmic derivative of the expansion factor as a function of r
function fdlexpansion(r, pexp)

Print parameters specific to expansion
subroutine expansion_print_parameters
```

`src/fexpand-MJ` for over-expansion according to Munro-Jackson (1981) [***ref***].

```
Module fexpand-MJ
Routines for Munro-Jackson (1981) expansion factor
Authors:
AV   Andrea Verdini
EB   Eric Buchlin
Modifications history:
19 Aug 05: AV Created
module fexpand
use types
use unit

Expansion factor as a function of r
non-dimensional unit
function fexpansion(r, pexp)

Logarithmic derivative of the expansion factor as a function of r
non-dimensional unit
function fdlexpansion(r, pexp)

Print parameters specific to expansion
subroutine expansion_print_parameters
```

4.2.6 Module `initialize`

This module, in the file `src/initialize.f90`, contains routines for initializing the fields, the atmosphere physical parameters, the wavenumbers in the perpendicular direction, and the random number generator.

```
Module initialize
Initialization of fields, of atmosphere, of wavenumbers and
of random numbers
Authors:
```

```
AV   Andrea Verdini
EB   Eric Buchlin
Modifications history:
22 Oct 05: EB Created (moved from shell-atm)
module initialize
use mpi
use types

Initialization of shells values
MPI: OK (no need, but be careful when changing initial field)
May be deprecated if we can write the initial fields in simstate.dat
by an external (IDL?) program
subroutine init_shells (zp, zm, k, p)
use diagnostics

Return in array k the values of the wavenumbers for each
index value in the perpendicular direction
Initialize random number generator
MPI: OK (done)
subroutine get_wavenumbers (k, p)
```

4.2.7 Module iodata

This module, in the file src/iodata.f90, contains routines for outputting the data during the simulation, including opening the files at the beginning of the simulation and closing them in the end.

The current alternatives are:

src/iodata-formatted.f90 for formatted data output

```
Module iodata
Output of data in files during the simulation,
opening and closing the files
Authors:
AV   Andrea Verdini
EB   Eric Buchlin
Modifications history:
22 Oct 05: EB Created (moved from shell-atm)
module iodata
use mpi
use types

Open files useful for output
MPI: OK (done)
use error codes?
subroutine output_open_files (p, outflags)

Output data in files for integrated quantities
MPI: OK (done)
subroutine output_data_a (zp, zm, k, p, tscales, outflags, outrecords, &
t, ti, lasttime)
use diagnostics
use iosimstate

Output data in files for non-integrated quantities
```

```
MPI: OK (done)
subroutine output_data_b (zp, zm, k, p, tscales, outflags, outrecords, &
t, ti, lasttime)
use diagnostics
use iosimstate

Close files used for output
MPI: OK (done)
subroutine output_close_files (p, outflags)
```

src/iodata-unformatted for unformatted data output

```
Module iodata
Output of data in files during the simulation,
opening and closing the files
Authors:
AV   Andrea Verdini
EB   Eric Buchlin
Modifications history:
22 Oct 05: EB Created (moved from shell-atm)
module iodata
use mpi
use types

Open files useful for output
MPI: OK (done)
use error codes?
subroutine output_open_files (p, outflags)

Output data in files for integrated quantities
MPI: OK (done)
subroutine output_data_a (zp, zm, k, p, tscales, outflags, outrecords, &
t, ti, lasttime)
use diagnostics
use iosimstate

Output data in files for non-integrated quantities
MPI: OK (done)
subroutine output_data_b (zp, zm, k, p, tscales, outflags, outrecords, &
t, ti, lasttime)
use diagnostics
use iosimstate

Close files used for output
MPI: OK (done)
subroutine output_close_files (p, outflags)
```

4.2.8 Module ioparams

This module, in the file src/ioparams.f90, contains routines for reading, displaying and outputting the parameters of the simulation (including the profiles of physical parameters of the atmosphere as computed at the beginning of the simulation).

```
Module ioparams
Input and output of parameters of simulation and of output
```

```
Authors:
AV   Andrea Verdini
EB   Eric Buchlin
Modifications history:
22 Oct 05: EB Created (moved from shell-atm)
module ioparams
use mpi
use types

Read parameters from file params.txt
MPI: OK (done)
subroutine read_parameters (p)

Print parameters on standard output
MPI: OK (done)
subroutine print_parameters (p)
use atmosphere
use boundary
use diagnostics
use fexpand
use initialize
use iodata
use iosimstate
use nonlin
use propagation
use types
use unit

Output data in files - Atmosphere profiles
Should be done only once a the beginning
MPI: OK
subroutine output_data_atm (p, outflags)
use atmosphere

Get output options
MPI: OK (done)
subroutine read_output_options (p, outflags)

Print output options on standard output
MPI: OK (done)
subroutine print_output_options (p, outflags)
```

4.2.9 Module `iosimstate`

This module, in the file `src/iosimstate.f90`, contains routines for reading and writing the simulation state, which can be used for analyses involving the whole fields and for restarting the simulation from the current time.

The current alternatives are:

`src/iodata-formatted.f90` for formatted data output

```
Module iosimstate-formatted
Input and output of simulation state, formatted data
Authors:
AV   Andrea Verdini
```

```
EB   Eric Buchlin
Modifications history:
22 Oct 05: EB Created (moved from shell-atm)
18 Nov 05: EB Made a module alternative (other is for unformatted output)
module iosimstate
use mpi
use types

Read state from which the simulation should continue
MPI: OK (done)
subroutine read_simstate (zp, zm, p, t)

Write state from which the simulation could continue
MPI: OK (done)
subroutine write_simstate (zp, zm, p, outflags, outrecords, t, ti, &
simstatefile)
```

`src/iodata-unformatted` for unformatted data output

```
Module iosimstate
Input and output of simulation state, unformatted data
Authors:
AV   Andrea Verdini
EB   Eric Buchlin
Modifications history:
22 Oct 05: EB Created (moved from shell-atm)
18 Nov 05: AV/EB Version for unformatted output (module alternative)
module iosimstate
use mpi
use types

Read state from which the simulation should continue
MPI: OK (done)
subroutine read_simstate (zp, zm, p, t)

Write state from which the simulation could continue
MPI: OK (done)
subroutine write_simstate (zp, zm, p, outflags, outrecords, t, ti, &
simstatefile)
```

4.2.10 Module `nonlin`

This module provides variables and routines to perform the integration of the non-linear term, with several alternatives. Each alternative should provide the variables:

- `module_nonlin`: identification of the module alternative

and the routines:

- `integrate_step_nonlin`: advance one time step of the the non-linear terms.

The current alternatives are:

`src/nonlin-goy.f90` for non-linear terms modelled through GOY shell-models (Giuliani and Carbone, 1998).

```
Module nonlin-goy
Integration of non-linear terms through GOY shell-models
Authors:
AV   Andrea Verdini
EB   Eric Buchlin
Modifications history:
24 Oct 05: EB Created (moved from shell-atm)
module nonlin
use types

Perform one step of 3rd order Runge-Kutta method integration
Tesi di laurea, Marco Ghiglione, p 38
Version for shell-loop: deals with 2D fields, so that Alfven-wave
propagation can be directly handled by the routine computing the
right handside of the differential equation (derivs2d)
MPI: OK (no need)
subroutine integrate_step_nonlin (zp, zm, k, p)

Right-hand-side of ODEs (=dzs/dt)
dzsdt: derivative (RHS of ODE) corresponding to zps
zps, zms: Elsasser-like spectra
dir: +1 if zps is z+ and zms z-, -1 if zps is z- and zms is z+
Version in 2D
MPI: OK (no need, as long as no propagation is done here)
subroutine derivs2d (dzsdt, zps, zms, k, p, dir)
use atmosphere
```

src/nonlin-goyb.f90 for non-linear terms modelled through GOY shell-models (Giuliani and Carbone, 1998) with a term for the Alfvén effect (Biskamp, 1994). The coefficient of this term is set by the `biskampa` variable inside this file.

```
Module nonlin-goyb
Integration of non-linear terms through GOY shell-models
with "Biskamp" term
Authors:
AV   Andrea Verdini
EB   Eric Buchlin
Modifications history:
24 Oct 05: EB Created (moved from shell-atm)
module nonlin
use types

Perform one step of 3rd order Runge-Kutta method integration
Tesi di laurea, Marco Ghiglione, p 38
Version for shell-loop: deals with 2D fields, so that Alfven-wave
propagation can be directly handled by the routine computing the
right handside of the differential equation (derivs2d)
MPI: OK (no need)
subroutine integrate_step_nonlin (zp, zm, k, p)

Right-hand-side of ODEs (=dzs/dt)
dzsdt: derivative (RHS of ODE) corresponding to zps
zps, zms: Elsasser-like spectra
dir: +1 if zps is z+ and zms z-, -1 if zps is z- and zms is z+
Version in 2D
```

```
MPI: OK (no need, as long as no propagation is done here)
subroutine derivs2d (dzsdt, zps, zms, k, p, dir)
use atmosphere
```

src/nonlin-nlsabra.f90 for non-linear terms modelled through non-local Sabra shell-models (a 2D version of Plunian and Stepanov, 2007).

```
Module nonlin-nlsabra
Integration of non-linear terms through 2D non-local shell-models
(adapted from the 3D non-local shell-models)
The non-locality exponent alpha is hard-coded in derivs2d()
Authors:
AV   Andrea Verdini
EB   Eric Buchlin
Modifications history:
03 Oct 12: EB Created (forked from nonlin-nlshell)
module nonlin
use types

Perform one step of 3rd order Runge-Kutta method integration
Tesi di laurea, Marco Ghiglione, p 38
Version for shell-loop: deals with 2D fields, so that Alfven-wave
propagation can be directly handled by the routine computing the
right handside of the differential equation (derivs2d)
MPI: OK (no need)
subroutine integrate_step_nonlin (zp, zm, k, p)

Right-hand-side of ODEs (=dzs/dt)
dzsdt: derivative (RHS of ODE) corresponding to zps
zps, zms: Elsasser-like spectra
dir: +1 if zps is z+ and zms z-, -1 if zps is z- and zms is z+
Version in 2D
MPI: OK (no need, as long as no propagation is done here)
subroutine derivs2d (dzsdt, zps, zms, k, p, dir)
use atmosphere
```

src/nonlin-none.f90 for no non-linear terms.

```
Module nonlin-none
No non-linear interactions
Authors:
AV   Andrea Verdini
EB   Eric Buchlin
Modifications history:
24 Oct 05: EB Created
module nonlin
use types

Empty routine (don't do integration of non-linear terms)
MPI: OK (no need)
subroutine integrate_step_nonlin (zp, zm, k, p)
```

4.2.11 Module `propagation`

This module provides variables and routines to compute the propagation of the Alfvén waves, with several alternatives. Each alternative should provide the variables:

- `module_propagation`: identification of the module alternative

and the routines:

- `propagate`: propagate the fields, with a given numerical scheme (except `zp` on first plane and `zm` on last plane which are computed by `force`

The current alternatives are:

`src/propagation-fromm.f90` for a propagation according to the Fromm numerical scheme (with Beam-Warming and Lax-Wendroff near the boundaries).

```
Module propagation-fromm
Variables and routines for Alfven wave propagation
With Fromm numerical scheme (inside the domaine)
Authors:
AV   Andrea Verdini
EB   Eric Buchlin
Modifications history:
07 Nov 05 EB Created (from numerical scheme by AV)
module propagation
use types

Alfven wave propagation, Fromm numerical scheme inside the box,
Beam-Warming when arriving at a boundary,
Lax-Wendroff when coming from a bounday.
zp on first plane and zm on last plane are _not_ computed (this is the
job of the force routine)
MPI: OK (done)
subroutine propagate (zp, zm, k, p, t)
use mpi
use atmosphere
```

`src/propagation-periodicfromm.f90` for a propagation according to the Fromm numerical scheme everywhere, assuming periodic boundary conditions (to be used with module `boundary-periodic`).

```
Module propagation-fromm
Variables and routines for Alfven wave propagation
With Fromm numerical scheme (inside the domaine)
Authors:
AV   Andrea Verdini
EB   Eric Buchlin
Modifications history:
07 Nov 05 EB Created (from numerical scheme by AV)
module propagation
use types

Alfven wave propagation, Fromm numerical scheme inside the box,
Beam-Warming when arriving at a boundary,
Lax-Wendroff when coming from a bounday.
zp on first plane and zm on last plane are _not_ computed (this is the
```

```
job of the force routine)
MPI: OK (done)
subroutine propagate (zp, zm, k, p, t)
use mpi
use atmosphere
```

4.2.12 Module `types`

This module, in the file `src/types.f90`, contains variables that can be shared by using the statement `use types` and user-defined types, as detailed in Sec. 4.1.1.

```
Module types
Definition of user-defined types
Authors:
AV    Andrea Verdini
EB    Eric Buchlin
Modifications history:
15 Jul 05: AV/EB: created

Random coefficients of forcing
type forcecoeffs

Parameters for the parallel run
type mpiparams

Parameters for the expansion function
type paramexp

Parameters of simulation (including parameters of the parallel run)
and for the expansion function
type params

What do we output, and at which frequency
type saveflags

Time scales as computed by the last call of check_time_scales
type timescales

Number of records saved for each type of output
type saverecords

Units (CGS) of variables
type units
```

4.2.13 Module `unit`

This module provides the numeric values (in CGS units) of the units used for adimensionalizing the variables of the model, with several alternatives. Units of length, time and mass should be provided, the units of the other variables should be consistent with these ones.

Each alternative should provide the variables:

- `module_units`: identification of the module alternative
- `un`: units, of user-defined type `units`
- `gg`: gravity acceleration at the base of the flux tube

- `Rsun`: radius of the Sun

Note: `gg` and `Rsun` are not used in all geometries. If they are not used, the units of the variables of the model don't need to be fixed in CGS units.

The current alternatives are:

`src/unit-open.f90`

```
Module unit-open
Units (CGS) for variables for open problem
Authors:
AV   Andrea Verdini
EB   Eric Buchlin
Modifications history:
30 Aug 05: AV/EB: created

length           time       mass
type(units), parameter :: un = units (6.961e10_dfloat, 1._dfloat, 1._dfloat)
type(units), parameter :: un = units (1._dfloat, 1._dfloat, 1._dfloat)
```

`src/unit-CGS.f90`

```
Module unit-open
Units (CGS) for variables for open problem
Authors:
AV   Andrea Verdini
EB   Eric Buchlin
Modifications history:
30 Aug 05: AV/EB: created

length           time       mass
type(units), parameter :: un = units (6.961e10_dfloat, 1._dfloat, 1._dfloat)
type(units), parameter :: un = units (1._dfloat, 1._dfloat, 1._dfloat)
```

4.2.14 External routines

`unlimitstack.c` The code is quite stack-intensive, we thus need to remove the system limitation on the size of the stack. This cannot be done by using a Unix command such as `limit` because this would be ineffective on the other processes launched by the MPI run. For this reason, we define the procedure `unlimitstack`, which is a wrapper around the Unix (SVr4, BSD 4.3) system call `setrlimit`.

4.3 Adding a module

- If adding a new alternative to an existing module, use the existing alternatives as a template, to be sure to provide the right variables and functions (with the same prototype).
- If you need to add a parameter to a function of the module, do so for all other module alternatives and update all calls to the function.
- If you add a new module, choose a name with no dash in it. Use a dash to separate the name of the module and the name of each alternative (if any). Update `print_parameters` (module `ioparams`) to display the module alternative.
- Update `src/Makefile`, adding your module and alternative (if relevant).

- Update `doc/forthtmldoc/Makefile` and `doc/manual/forttexdocs/Makefile` with your module.
- Update `doc/manual.tex`.

4.4 Parallelization

The total number of planes `p%tnz` is distributed over the `p%mpi%np` processors: if `p%tnz` = `p%mpi%np`·⌊`p%tnz`/`p%mpi%np`⌋+`excessplanes` with `excessplanes` < `p%mpi%np`, the first `excessplanes` processors receive `p%nz` = ⌊`p%tnz`/`p%mpi%np`⌋+1 planes and the other ones receive `p%nz` = ⌊`p%tnz`/`p%mpi%np`⌋ planes. This is done in module `ioparams`.

There is no need for an extension of the domain towards neighboring processors for the fields `zp` and `zm`. It is indeed sufficient that there is an extension of the domain for the old values of the fields, `oldzp` and `oldzm`, kept in `propagate` (module `propagation`).

Thanks to this, a change in the numerical scheme for Alfvén wave propagation (which could require more or less extended grid points than another scheme), affects only variables local to `propagate`.

4.4.1 Performance test with interactive runs

To do a performance test of the parallelization in interactive mode on `berenice.arcetri.astro.it`:

- install the code in a subdirectory of `runs/` and compile it
- copy `sys/perftest.pl` and edit it to choose the number of planes and processors to test
- run `perl perftest.pl`, this produces a file `perftest.txt`
- use the IDL routine `perftest_an` to analyze the data contained in `perftest.txt`

4.5 Bugs

- With Intel Fortan 8.0, the allocation of arrays fails sometimes with an error code indicating that the array is already allocated (which is not the case). Systematically veryfying the error code of `allocate` avoids the bug, which seems to be a compiler bug.

4.6 Wishlist

- Use a public versionning system (using SVN locally)
- Append output files instead of replacing them (done)
- Website automatic building (done)
- Selection of different sets of parameters (done)
- Handling non-equidistant planes (done)
- Including a wind (advection by parallel velocity) (done)
- Make the code more versatile (other outputs, other fields, other physics)
- Automatic unitary tests of module alternatives
- Find a fancy acronym
- HDF5 output
- Python analysis routines
- Automatic selection of platform at compile time (instead of installation time)
- Better random number generator initialization (something deterministic but different for each run?)

5 A few tricks: how to...

5.1 Continue a run

To continue a run, you usually copy all files to a new subdirectory of `runs`, so that the output from the first run is not overwritten by the second run. But it is not very efficient to copy your big output files from the first run, which need to be deleted afterwards anyway. So, in `runs`, instead of using `cp -R firstrun secondrun`, you can exclude the output files (which are the files beginning by the letter `o` by using `mkdir secondrun; cp firstrun/[^o]* secondrun`.

Then edit `params.txt` to indicate the end time of the new run and to say that you want to restart from the existing `simstate.dat`, and go!

It can be a good practice to keep a copy of the original simulation state (the final `simstate.dat` produced by the first run) in a file `simstate.dat_0`. This would allow to redo the second run without needing to think about what run was the first run (if it still exists!).

5.2 Continue a run with increased number of planes

After having copied the files (Sec. 5.1), to increase the number of planes before continuing a run, move `simstate.dat` to `simstate.dat_0`. This will be conserved as the simulation state before interpolation. Then take the IDL procedure `idl/shellinterp.pro` as an example of using the IDL procedures `idl/sa_params_get.pro` and `idl/sa_params_write.pro`, together with the IDL intrisic procedure `interpol`, to easily perform the interpolation necessary to get a starting simulation state with more planes, that will be produced in `simstate.dat_1`.

You should also edit `params.txt` to update the number of planes and the distance between planes. Then copy `simstate.dat_1` to `simstate.dat`, and go!

A Authors and contributors

Éric Buchlin	eric.buchlin@ias.u-psud.fr	Institut d'Astrophysique Spatiale, France
Andrea Verdini	verdini@arcetri.astro.it	Università di Firenze, Italy
Marco Velli	velli@arcetri.astro.it	Università di Firenze, Italy

B License agreement

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License[5] as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

We also ask that that publications and presentations making use of this code give reference to Buchlin and Velli (2007) or Verdini *et al.* (2009) (depending on the modules you use) and contain the URL of the code documentation (http://eric.buchlin.org/research#shellatm).

References

Biskamp, D. (1994). Cascade models for magnetohydrodynamic turbulence. *Phys. Rev. E*, **50**, 2702–2711.

Boffetta, G., Carbone, V., Giuliani, P., Veltri, P., and Vulpiani, A. (1999). Power Laws in Solar Flares: Self-Organized Criticality or Turbulence? *Phys. Rev. Lett.*, **83**, 4662–4665.

Buchlin, E. and Velli, M. (2007). Shell Models of RMHD Turbulence and the Heating of Solar Coronal Loops. *Astrophys. J.*, **662**, 701–714.

[5] http://www.gnu.org/copyleft/gpl.html

Dmitruk, P., Milano, L. J., and Matthaeus, W. H. (2001). Wave-driven Turbulent Coronal Heating in Open Field Line Regions: Nonlinear Phenomenological Model. *Astrophys. J.*, **548**, 482–491.

Giuliani, P. and Carbone, V. (1998). A note on shell models for MHD turbulence. *Europhys. Lett.*, **43**, 527–532.

Giuliani, P., Jensen, M. H., and Yakhot, V. (2002). Critical "dimension" in shell model turbulence. *Phys. Rev. E*, **65**(3), 036305–036309.

Gledzer, E. B. (1973). System of hydrodynamic type admitting two quadratic integrals of motion. *Sov. Phys. Dokl.*, **18**, 216–217.

Gloaguen, C., Léorat, J., Pouquet, A., and Grappin, R. (1985). A scalar model for MHD turbulence. *Physica D*, **17**, 154–182.

Plunian, F. and Stepanov, R. (2007). A non-local shell model of hydrodynamic and magnetohydrodynamic turbulence. *New Journal of Physics*, **9**, 294–+.

Verdini, A., Velli, M., and Buchlin, E. (2009). Turbulence in the Sub-Alfvénic Solar Wind Driven by Reflection of Low-Frequency Alfvén Waves. *Astrophys. J.*, **700**, L39–L42.

Yamada, M. and Ohkitani, K. (1987). Lyapunov spectrum of a chaotic model of three-dimensional turbulence. *J. Phys. Soc. Japan*, **56**, 4210–4213.

Yamada, M. and Ohkitani, K. (1988a). Lyapunov spectrum of a model of two-dimensional turbulence. *Phys. Rev. Lett.*, **60**, 983–986.

Yamada, M. and Ohkitani, K. (1988b). The Inertial Subrange and Non-Positive Lyapunov Exponents in Fully-Developed Turbulence. *Progr. Theor. Phys.*, **79**, 1265–1268.